Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data

Reynold Cheng   Yuni Xia   Sunil Prabhakar   Rahul Shah   Jeffrey Scott Vitter

Department of Computer Sciences, Purdue University
West Lafayette
IN 47907-1398, USA
{ckcheng,xia,sunil,rahul,jsv}@cs.purdue.edu

Abstract

It is infeasible for a sensor database to contain the exact value of each sensor at all points in time. This uncertainty is inherent in these systems due to measurement and sampling errors, and resource limitations. In order to avoid drawing erroneous conclusions based upon stale data, the use of uncertainty intervals that model each data item as a range and associated probability density function (pdf) rather than a single value has recently been proposed. Querying these uncertain data introduces imprecision into answers, in the form of probability values that specify the likeliness the answer satisfies the query. These queries are more expensive to evaluate than their traditional counterparts but are guaranteed to be correct and more informative due to the probabilities accompanying the answers. Although the answer probabilities are useful, for many applications, it is only necessary to know whether the probability exceeds a given threshold – we term these *Probabilistic Threshold Queries* (PTQ). In this paper we address the efficient computation of these types of queries.

In particular, we develop two index structures and associated algorithms to efficiently answer PTQs. The first index scheme is based on the idea of augmenting uncertainty information to an R-tree. We establish the difficulty of this problem by mapping one-dimensional intervals to a two-dimensional space, and show that the problem of interval indexing with probabilities is significantly harder than interval indexing which is considered a well-studied problem. To overcome the limitations of this R-tree based structure, we apply a technique we call *variance-based clustering*, where data points with similar degrees of uncertainty are clustered together. Our extensive index structure can answer the queries for various kinds of uncertainty pdfs, in an almost optimal sense. We conduct experiments to validate the superior performance of both indexing schemes.

1   Introduction

Uncertainty is a common problem faced by sensor databases that interact with external environments. Consider a database system which stores and monitors current pressure, with pressure sensors being deployed in the venue being investigated. Due to resource limitations such as battery power of sensors and network bandwidth, it is often infeasible for a sensor database to contain the exact value of each sensor at all points in time. In particular, since pressure is a continuously changing entity, the system only receives old samples of pressure values. The situation is not helped by the fact that sensor data may not arrive at the system on time, and may even be lost due to network problems. The measurement error incurred by the sensor while measuring the value further aggravates the problem.

In general, uncertainty occurs in any database that attempts to model and capture the state of the physical world, where entities being monitored such as pressure, temperature, locations of moving objects, are constantly changing. As pointed out in [7], if the received (stale) sensor value is directly used to answer queries, erroneous answers may result. In order to alleviate this problem, the idea of incorporating uncertainty information into the sensor data has been pro-

posed recently. Instead of storing single values, each item is modeled as a range of possible values, associated with a probability density function (pdf) [7].

With the notion of uncertainty, querying on data generates imprecise, rather than exact answers. In these *probabilistic queries*, answers are augmented with probability values that specify the likelihood that they satisfy the query. As an example, suppose there are ten sensors, namely $s_1, s_2, \ldots, s_{10}$ that monitor the temperature values in different offices of a building. Without considering uncertainty, a query that inquires which sensor has a temperature value over 30^oF may yield the answer $\{s_1, s_3, s_9\}$. On the other hand, its probabilistic counterpart can generate $\{(s_1, 0.9), (s_3, 0.7), (s_8, 0.6), (s_9, 0.1)\}$ as an imprecise answer. Here we can observe that sensor s_1 has a very high probability of producing a temperature value over 30^oF, while s_9 only has a marginal chance of satisfying the query. A probabilistic query thus allows us to see the difference in the likelihood of each answer satisfying a query. We can also see that s_8 is a new member of the answer to the probabilistic query, which is *not* in the answer set of the query that does not consider data uncertainty. This is probably because the perceived value of s_8 received by the database is less than 30^oF, but in fact its current actual value can be higher than 30^oF with a non-trivial chance (0.6). A probabilistic query is thus able to produce a more accurate and informative answer than a traditional query.

Despite these advantages over their traditional counterparts, probabilistic queries suffer from a serious problem: they are much more expensive to evaluate. While traditional queries require only exact and single data inputs, probabilistic queries must manage uncertainty information, including intervals and pdfs. In particular, probability values augmented to query answers can be obtained only after costly integration operations are performed. However, it should be noted that although answer probabilities are useful, in practice for many queries, it is only necessary to know whether the probability of answer exceeds a given threshold – we term these *Probabilistic Threshold Queries* (PTQ). A PTQ version for the previous example can be "return the ids of the sensors that have values over 30^oF with a probability of greater than or equal to 0.7", in which case the answer $\{s_1, s_3\}$ is produced.

By exploiting the probability threshold requirement on a probabilistic query, we propose efficient searching techniques. We investigate how an index data structure and its associated algorithms are developed for imprecise data. Two index structures are developed to answer PTQs efficiently. The first index scheme is based on the novel idea of augmenting uncertainty information to an R-tree, where the number of I/Os and integration operations are reduced significantly while the index is still being visited. Although this idea is simple to implement, it is far from being optimal. We then change our focus to study the theoretical complexity of indexing uncertainty, and argue that there is no formerly known optimal solution that is applicable to this problem. By mapping one-dimensional intervals to a two-dimensional space, we illustrate that the problem of indexing uncertainty with probabilities is significantly harder than interval indexing, which is considered a well-studied problem.

Based on the interpretation of theoretical studies, we develop a technique called *variance-based clustering*, in order to overcome the limitations of the uncertainty-information-augmented R-tree structure. In this indexing scheme, data points with similar degrees of uncertainty (e.g., mean and standard deviation) are clustered together. The final extensive index is an R-tree based index, augmented with uncertainty information, and enhanced with the variance-based-clustering technique. This index can answer the queries for various kinds of uncertainty pdfs, in an almost optimal sense. The results are verified by an extensive experimental evaluation.

As a summary of our contributions, we propose two structures to index uncertain data for PTQs. The first index, called *PTI*, augments uncertain information to internal nodes so that more search paths can be pruned while the index is being visited. This index forms the basis of a more extensive scheme, where intervals with similar variance values are clustered together. We also show that with a fixed probability threshold, querying uncertainty intervals with uniform pdf can be answered in optimal time, and establish a theoretical foundation of the problem. We also perform extensive experiments to compare our proposed schemes with an R-tree.

The rest of this paper is organized as follows. In Section 2 we formally define the uncertainty model and probabilistic threshold queries. Section 3 presents a simple index with uncertainty information augmented to evaluate a PTQ. We further establish the theoretical difficulty of the problem in 4. An extensive framework for evaluating PTQ is presented in 5. We present our experimental results in 6. Section 7 discusses related work and Section 8 concludes the paper.

2 Data Uncertainty and Probabilistic Queries

In [7][9], a data representation scheme known as *probabilistic uncertainty model* was proposed. The model requires that at the time of query execution, the range of possible values of the attribute of interest, and their distributions, are known. For notational convenience, we assume that a real-valued attribute a of a set of database objects T is queried. The ith object of T is named T_i, and the value of a for T_i is called $T_i.a$ $(i = 1, \ldots, |T|)$, where $T_i.a$ is treated as a continuous random variable. The *probabilistic uncertainty* of $T_i.a$

consists of two components:

Definition 1 *An **uncertainty interval** of $T_i.a$, denoted by U_i, is an interval $[L_i, R_i]$ where $L_i, R_i \in \Re$, and the conditions $R_i \geq L_i$ and $T_i.a \in U_i$ always hold.*

Definition 2 *An **uncertainty pdf** of $T_i.a$, denoted by $f_i(x)$, is a pdf of $T_i.a$, such that $f_i(x) = 0$ if $x \notin U_i$.*

This simple model provides flexibility where the exact model of uncertainty is determined by application-dependent assumptions. A simple example is the modeling of sensor measurement uncertainty, where each U_i is an error range containing the mean value, and $f_i(x)$ is a normal distribution. Another example is the modeling of one-dimensional moving objects based on [20], where at any point in time, the actual location is within a certain bound, d, of its last reported location value. If the actual location changes further than d, then the sensor reports its new location value to the database and possibly changes d. In this case, U_i contains all the values within a distance of d from its last reported value. For $f_i(x)$, one may assume that $T_i.a$ is uniformly distributed, i.e., $f_i(x) = 1/[R_i - L_i]$ for $T_i.a \in U_i$. Treating $f_i(x)$ as a uniform pdf models the scenario where $T_i.a$ has an *equal* chance of locating anywhere in U_i. Due to its simplicity, a uniform distribution facilitates ease of analysis and efficient index design, as illustrated in subsequent sections.

Alternatively, one may perform an estimation of the pdf based on time-series analysis, the discussion of which is beyond the scope of this paper. Interested readers are referred to [5] for details. Also notice that we limit our discussion of uncertainty to interval data. A comprehensive discussion of different types of uncertainty can be found in [21].

A *probabilistic threshold query* (PTQ), proposed in [9], is a variant of probabilistic query, where only answers with probability values over a certain threshold p are returned. The PTQ that we study specifically in this paper is defined formally below.

Definition 3 Probabilistic Threshold Query *(PTQ) Given a closed interval $[a, b]$, where $a, b \in \Re$ and $a \leq b$, a PTQ returns a set of tuples T_i, such that the probability $T_i.a$ is inside $[a, b]$, denoted by p_i, is greater than or equal to p, where $0 < p \leq 1$.*

Simply speaking, a PTQ can be treated as a range query, operating on probabilistic uncertainty information, and returns items whose probabilities of satisfying the query exceed p.

3 A Simple Uncertainty Index

A naive method to evaluate a PTQ is to first retrieve all T_i's, whose uncertainty intervals have some overlapping with $[a, b]$, into a set S. Each T_i in S is then evaluated for their probability of satisfying the PTQ with the following operation:

$$p_i = \int_{OI} f_i(x)dx \qquad (1)$$

where p_i is the probability that T_i satisfies the PTQ, and OI is the interval of overlap between $[a, b]$ and U_i. The answer only includes T_i's whose p_i's are larger than p.

Two problems can be seen from this approach. First, how can we find the elements of S i.e., U_i's that overlap with $[a, b]$? It can be very inefficient if each item T_i is retrieved from a large database and tested against $[a, b]$. A typical solution is to build an index structure over U_i's (which are intervals) and apply a range search of $[a, b]$ over the index. This problem is known as the interval indexing problem, and has been well studied [17][15].

The second problem is that the probability of each element in S needs to be evaluated with Equation 1. This can be a computationally expensive operation. Notice that the bottleneck incurred in this step is independent of whether we use an interval index or not. In particular, the interval index does not help much if many items overlap with $[a, b]$, but most have probability less than p. In this situation, we still need to spend a lot of time to compute the probability values for a vast number of items, only to find that they do not satisfy the PTQ after all.

3.1 Probability Threshold Indexing

The above problems illustrate the inefficiency of using an interval index to answer a PTQ. While the range search is being performed in the interval index, only uncertainty intervals are used for pruning out intervals which do not intersect $[a, b]$. Another piece of important uncertainty information, namely the uncertainty pdf, has not been utilized at all in this searching-and-pruning process. As a result, a large number of items may overlap with $[a, b]$, while in fact only a small fraction of them contribute to the results of PTQ.

Our goal is to redesign index structures so that probabilistic uncertainty information is fully utilized during an index search. This structure, called *Probability Threshold Indexing* (PTI), is based on the modification of a one-dimensional R-tree, where probability information is augmented to its internal nodes to facilitate pruning. To illustrate our idea, let us review briefly how a range query is performed on an R-tree. Starting from the root node, the query interval $[a, b]$ is compared with the maximum bounding rectangle (MBR) of each child in the node. Only children with MBRs that overlap with $[a, b]$ are further followed. We thus save the effort of retrieving nodes whose MBRs do not overlap $[a, b]$. We can generalize this idea by constructing *tighter* bounds (that we call x-bounds) than
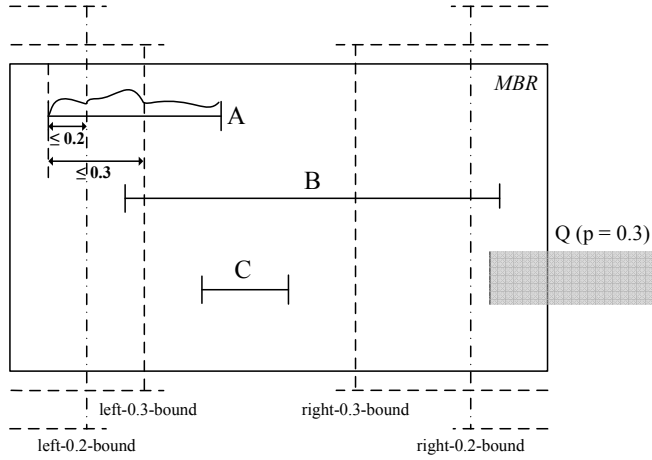
Figure 1: Inside an MBR M_j, with a 0.2-bound and 0.3-bound. A PTQ named Q is shown as an interval.

the MBR in each node, by using uncertainty information of intervals, so as to further reduce the chance of examining the children of the node. Let M_j denote the MBR/uncertainty interval represented by the jth node of an R-tree, ordered by a pre-order traversal. Then the x-bound of M_j is defined as follows.

Definition 4 *An* **x-bound** *of an MBR/uncertainty interval M_j is a pair of lines, namely left-x-bound (denoted by $M_j.lb(x)$) and right-x-bound (denoted by $M_j.rb(x)$). Every item $T_i.a$ contained in M_j can only have a probability of at most x (where $0 \leq x \leq 1$) both on the left of the left-x-bound and on the right of the right-x-bound. That is to say, if $L_i \leq M_j.lb(x)$ and $R_i \geq M_j.rb(x)$, then the following must hold: $\int_{L_i}^{M_j.lb(x)} f_i(y)dy \leq x$ and $\int_{M_j.rb(x)}^{R_i} f_i(y)dy \leq x$.*

Using the definition of an x-bound, the MBR of an internal node can be viewed as a 0-bound, since it guarantees all intervals in the node are contained in it with probability one i.e., no interval lies beyond the 0-bound. Figure 1 illustrates three children MBRs (A,B,C), in the form of one-dimensional intervals, contained in a larger MBR M_j. A 0.2-bound and a 0.3-bound for M_j are also shown.

As Figure 1 shows, an x-bound is a pair of lines where at most a fraction of x of each interval in the MBR cross either of them. The uncertainty pdf of A is shown, where we can see that $\int_{L_A}^{M_j.lb(0.2)} f_i(x)dx \leq 0.2$, and $\int_{L_A}^{M_j.lb(0.3)} f_i(x)dx \leq 0.3$. For interval B, the constraint on the right-0.3-bound is $\int_{M_j.rb(0.3)}^{R_B} f_i(x)dx \leq 0.3$. Interval C does not cross either the 0.2-bound and the 0.3-bound, so it satisfies the constraints of both x-bounds. Furthermore, we require an x-bound to be unique, where the left-x-bound and right-x-bound are pushed towards the center of the MBR as much as possible, without violating their definitions.

The whole purpose of storing the information of the x-bound in an R-tree node is to avoid investigating the contents of a node. If we can avoid this probing, a considerable amount of I/Os can be saved. Furthermore, we do not need to compute the probability values of those intervals, which cannot satisfy the query anyway. To illustrate how this idea works, let us look at Figure 1 again. Here a range query Q, represented as an interval, is tested against the internal node. Without the aid of the x-bound, Q has to (i) examine which MBR (i.e., A, B, or C) overlaps with Q's interval, (ii) for the qualified MBRs (B in this example), further retrieve the node pointed by B until the leaf level is reached, and (iii) compute the probability of the interval in the leaf level.

The presence of the x-bound allows us to decide with ease whether an internal node contains any qualifying MBRs, without further probing into the subtrees of this node. In this example, we first test Q's range against the left-0.2-bound and the right-0.2-bound. As shown in Figure 1, it intersects none of these bounds. In particular, although Q overlaps the MBR, its overlapping region is somewhere between the right-0.2-bound and the right boundary of M_j's MBR. Recall that a 0.2-bound allows at most an accumulated pdf of 0.2 of any interval in an MBR. This implies that the portion of the intervals (interval B) that passes through the 0.2-bound cannot exceed a probability of 0.2. Therefore, the probability of intervals in the MBR that overlap the range of Q cannot be larger than 0.2. Assume Q has a probability threshold of 0.3 i.e., Q only accepts intervals with an overlapping probability of at least 0.3. Then we can be certain that *none* of the intervals in the MBR satisfies Q, without further probing the subtrees of this node. Compared with the case where no x-bounds are implanted, this represents a significant saving in terms of the number of I/Os and computation time.

In general, given an x-bound of a MBR M_j, and a PTQ with interval $[a, b]$ and probability threshold p, we can eliminate M_j from further examination if the following two conditions hold:

1. $[a, b]$ does not intersect left-x-bound or right-x-bound of M_j i.e., either $b < M_j.lb(x)$ or $a > M_j.rb(x)$ is true, and

2. $p \geq x$

If no x-bound in M_j satisfies these two conditions, the checking of intersections with M_j is resumed, where the contents of the node represented by M_j are loaded, and the range searching process is done in the same manner as for an R-tree.

3.2 Implementation of PTI

Figure 2 illustrates an implementation of PTI. Its framework is the same as R-tree, where each internal
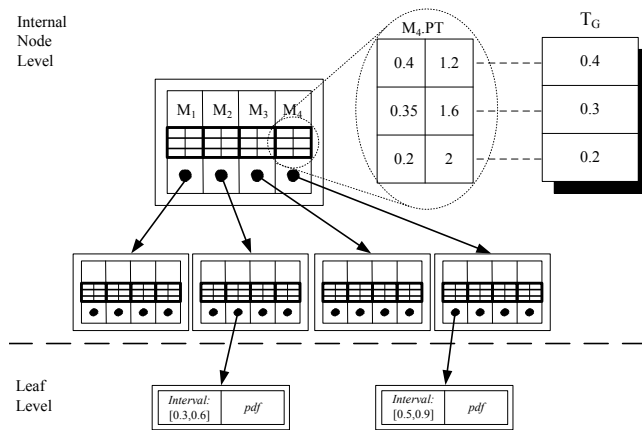
Figure 2: Structure of PTI

node consists of children MBRs and their corresponding pointers. In addition, each child M_j consists of a table, $M_j.PT$, that contains information of its x-bounds. Each entry of $M_j.PT$ is a tuple of the form <left-x-bound, right-x-bound>. Further, a global table called T_G is defined, which contains the values of x for x-bounds. The i-th entry of $M_j.PT$ contains the x-bound whose value of x is stored in the i-th entry of T_G. The data items being indexed are essentially uncertainty intervals and pdfs.

To insert $T_i.a$, we first compute its x-bounds, corresponding to the values of x in T_G. Then we insert $T_i.a$ to the PTI, using a similar procedure as inserting an interval to an R-tree. The main difference is that the x-bounds of the intermediate nodes being traversed during insertion need to be expanded appropriately. In particular, the left-x-bound of an internal node needs to be replaced by the corresponding left-x-bound of $T_i.a$ if the former value is larger than the latter. The right-x-bounds are expanded analogously. Finally, the x-bound information computed for $T_i.a$ is copied to PT of the node that directly points to $T_i.a$.

Removing an object follows a similar procedure of the R-tree. Again, we need to take care of the update issues of x-bounds. We observe that if an MBR M_j is to be deleted, then the left-x-bound of the parent node that points to M_j has to be shrinked to the minimum of left-x-bound of all MBRs in the same node as M_j. The right-x-bound of the parent node is adjusted in a similar manner. We therefore need to keep parent pointers in each node. To update the x-bounds, beginning from the leaf node that contains the interval of interest, the changes to x-bounds are propagated until the root is reached.

Although the fan-out of a PTI node is lower than an R-tree node because each node contains less space to store MBRs (assume the node size is fixed), the fan-out only logarithmically affects the height of the tree. Hence, in most cases this results in an increase in height by an additive constant, which only has a minor effect on PTI's performance. Indeed, its performance

illustrates significant improvements over R-tree, as observed in our experimental results (Section 6).

However, PTI by itself is not an optimal solution, because it cannot avoid the problem that an R-tree faces – if the data source consists of both large and small intervals, a lot of smaller intervals will reside in the same leaf node as the large intervals. The search time is increased unnecessarily, because a range search may have to go through many large MBRs consisting of large intervals. The major cause of this problem is that the insertion mechanism of the R-tree does not differentiate between large and small intervals. We investigate this problem in subsequent sections, and develop an extensive framework to tackle this shortcoming. The framework employs the idea of PTI as well. In some cases, the framework is even able to eliminate the extra overhead of PTI altogether, by computing the probability threshold information "on the fly".

4 Theoretical Implications

Any index is considered theoretically efficient if it achieves provably logarithmic update and query times while using a linear amount of space. We will discuss the difficulty of the PTQ problem as compared with other known problems in computational geometry. Interval indexing [14, 3, 2] is considered a well-studied problem and theoretically efficient indexes exist. However, we show here that interval indexing coupled with pdfs is significantly more complex than interval indexing. On the other hand, theoretically efficient index structures for PTQ are possible only when the threshold p is apriori fixed constant for all the queries.

In this section, we will mainly focus on PTQ assuming the pdf in each interval is uniform. That is, if a query specifies 80% threshold, then any interval satisfying this query has at least 80% of its length within the query range. We call this the PTQU problem. In this section, we want to show that PTQ is a hard problem to be provably solved even with uniform pdfs. However, good heuristics can be used for PTQU and they can be extended to PTQs when pdfs are arbitrary using the idea of PTI in the previous section. As a side note, we also show that a provably good index for PTQU can exist if the threshold of probability p is a fixed constant.

4.1 2D mapping of intervals

We first explore PTQUs when the intervals are indexed as points in two dimensional space [14, 3]. Here, each interval $[x, y]$ is mapped to a point (x, y) in 2D. Note that, for all intervals, $x < y$ and hence these points all lie in the region above (and to the left of) the line $x = y$. Figure 3(a) gives the illustration. A stabbing query is a particular kind of query associated with the notion of intervals. Given a point c, a stabbing query reports all the intervals containing point c. A stabbing query [3] for point c is converted to a two-sided orthogonal
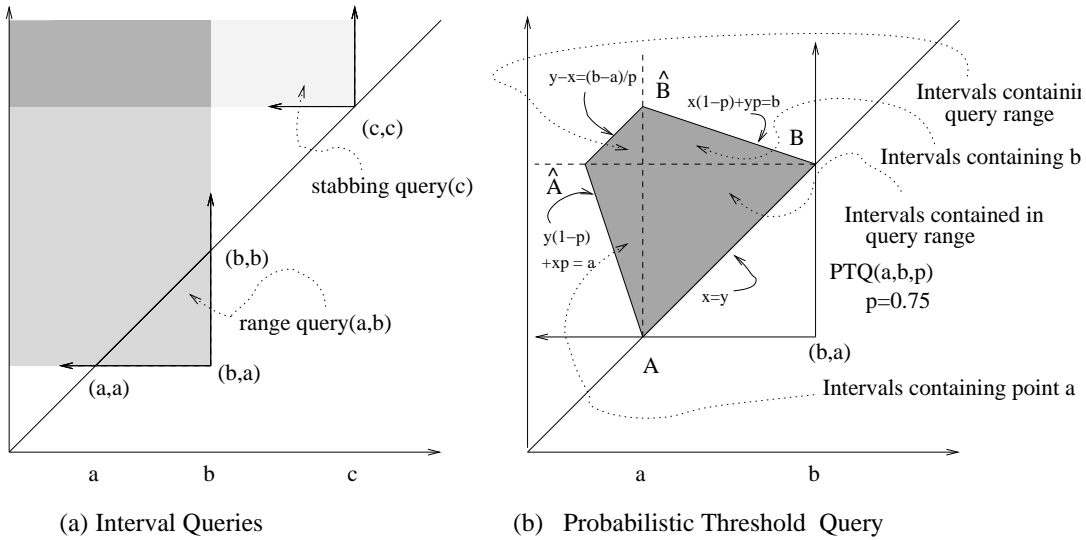
Figure 3: Probabilistic Threshold Queries with Uniform pdf

query originating at point (c, c) in this mapping. A range query (a, b) is just a union of the stabbing queries for all the points from a to b. This is same as a two-sided orthogonal query originating at point (b, a).

A PTQU (a, b, p) where $0 < p \leq 1$ now becomes a 3-sided trapezoidal query as shown in Figure 3(b). To see this, consider any point (x, y) (i.e. interval $[x, y]$) that satisfies PTQU. There are four cases:

$x \leq a < b \leq y$: In this case the query lies within the interval. All we require is that the query covers a sufficient length of the interval. That is $b - a \geq p(y - x)$. That means point (x, y) is in the region below the line $y - x = (b - a)/p$. This line has slope 1.

$x \leq a < y \leq b$: In this case the query region is on the right of the interval. The amount of overlap is given by $y - a$. This condition translates to $y(1 - p) + xp \geq a$. That is the region above the line $y(1 - p) + xp = a$ which has slope $-p/(1 - p)$.

$a \leq x < b \leq y$: In this case is the query region is on the left of the interval. This is given by the region $x(1 - p) + yp \leq b$. The separating line has slope $-(1 - p)/p$.

$a < x < y < b$: In this case, the entire interval lies within the query and hence it satisfies the PTQU for any p.

Thus, the query satisfying region is given by the intersection of the three regions (first three) above. This becomes an isosceles trapezoid region. The fourth side of the region given by line $x = y$ can be essentially considered redundant since there are no points below (or to the right) of this line. We will call this as an open side of the trapezoid. Thus, PTQU becomes a 3-sided trapezoidal query.

As p approaches 0, this becomes a range intersection query i.e. the slopes of the lines in the second and third cases become zero and infinity respectively. The first constraint becomes redundant. This is the same as a two-sided orthogonal query. At $p = 1$, the trapezoid becomes a right-angled triangle and the query becomes a containment query. At $p = 0.5$ the trapezoid becomes a square. For $p < 0.5$ the close side (as given by first constraint) of the trapezoid is bigger than the open side (on line $x = y$) and for $p > 0.5$ the closed side is smaller than the open side. See Figure 5.

4.2 Relation of PTQU to other well known problems

To establish the difficulty of the problem we will relate PTQU to two well known problems, namely simplex queries in 2D and half-space queries in 2D. First we define these problems and then show that PTQU lies between these two problems in terms of its hardness. This indicates that a provably good indexing may not be possible for PTQU. On the other hand we shall show in the next subsection that if the threshold for PTQU is fixed for all the queries, then provably good query times can be achieved.

Problem 1 Half-space queries in 2D (HQ2D): *Given a dynamic set of points in 2D, report the set of points which satisfy a query given by a linear constraint $ax + by \geq c$ where a, b, c are real numbers.*

Problem 2 Simplex queries in 2D (SQ2D): *Given a dynamic set of points in 2D, report the set of points which satisfy a query given by a constant number of linear constraints $a_i x + b_i y \geq c_i$ where i goes from 1 to a constant j and a_i, b_i, c_i are real numbers.*

It is easy to see that PTQU is a special case of Simplex queries. Also, we can establish that PTQU

(a) 2–sided orthogonal query   (b) 3–sided orthogonal query   (c) Fixed angle (slope) wedge queries   (d) Fixed angle 3–sided parallelogram queries
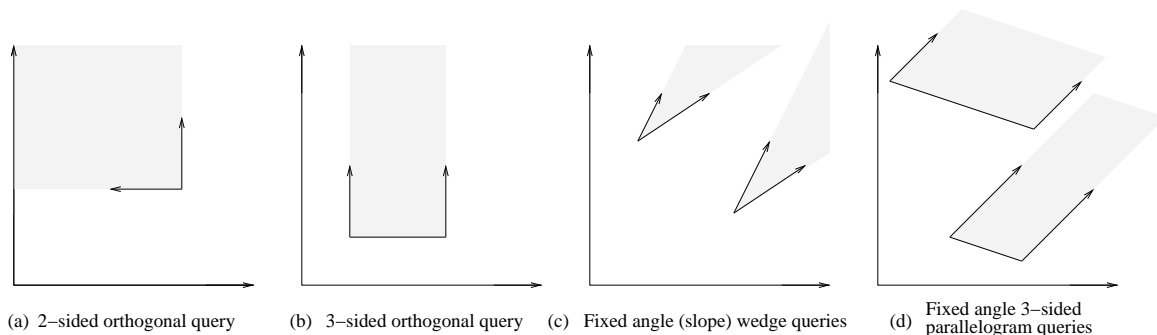
Figure 4: Wedge queries and parallelogram queries with fixed slopes

is at least as hard as half-space queries. Let n be the number of points and k be the number of points satisfying a query. We state the following lemma. The proof is skipped for conciseness.

Lemma 1 *If PTQU for n intervals can be answered in time t with update times u and using space s, then HQ2D can be answered in $O(t)$ time, with update time $O(u)$ and in $O(s)$ space.*

☐

Here, we consider problems (HQ2D and SQ2D) which report all the points satisfying the query. These are called reporting versions of the problem. Lower bounds exist for these problems in the algebraic model [12, 6]. Data structures which use a linear amount of storage for HQ2D require at least $\Omega(n^{1/3})$ for query and update operations on average[12, 4]. For SQ2D, this lower bound is $\Omega(\sqrt{n})$ [6]. The best known data structures can answer SQ2D queries in $\sqrt{n}\log n$, which is considered almost tight against the lower bound. Half-space range searching, is one of the exceptions in the class of geometric range searching problems where lower bounds from algebraic models do not apply to the reporting version of the problem (i.e. HQ2D as we have defined). See the surveys on Geometric range searching by [18, 11] for more details. However, the best known data structure for HQ2D can answer reporting queries in time $O(n^\epsilon + k)$ using linear storage [1]. If the storage is allowed to be super-linear i.e. $O(n^{1+\epsilon})$, then optimal reporting time of $O(\log n + k)$ can be achieved [10]. Note that ϵ can be made arbitrarily small but this increases the constants hidden in the big-O notation. This implies that we can not hope for a linear space index which gives provably good query times (i.e. $O(\log n + k)$).

However, the above bounds are for the worst case performance of the data structure. In general, when points in 2D are uniformly distributed (and not pathologically arranged to force the worst case), any space partitioning data structure like R-tree gives reasonably good query times for polygonal queries (i.e. SQ2D). Goldstein et al. [13] use R-trees to answer SQ2D queries. Although worst case bounds can not be proven, practically the index works fairly well.

Motivated by [13], we use R-tree in 2D to answer PTQU. For a general pdf (not necessarily uniform) we develop a heuristic based on this idea to answer PTQ. More details are provided in Section 5.

4.3 PTQU with fixed threshold

Here we show that PTQU can be answered provably efficiently when the threshold value p is fixed for all queries and p is not close to 0. However, we wish to note that these results are for theoretical interest only. We can use the data structures which handle 2-sided and 3-sided orthogonal queries [2] (see Figure 4(a,b)). First we establish the following lemma when 2-sided or 3-sided queries are not orthogonal but consist of line segments with fixed slopes (see Figure 4(c,d)). Also, in external memory where the cost model is number of I/Os, with block size B, a 2-sided angular query (also called wedge queries) is a query specified by an angle (i.e. two rays) and it reports all the points in the interior of the angle. A 3-sided parallelogram query is a generalization of 3-sided orthogonal queries, where there are two parallel sides and one closed side, and each of these can be at an arbitrary inclination (not just orthogonal).

Lemma 2 *For n points in two dimensional space, 2-sided wedge queries where the slopes of both the sides are fixed for all queries and 3-sided parallelogram queries where two independent slopes involved are fixed for all queries can be answered in $O(\log n + k)$ time. Also, in external memory where block size for I/Os is B, these queries can be answered in $O(\log_B n + k/B)$ I/Os.*

Proof : (sketch) Since the slopes of lines involved are fixed for all the queries, we use a linear transformation to align the directions of the axes along these slopes. Now, these queries simply become 2-sided and 3-sided orthogonal queries. Known structures from [2] can be used to answer them. ☐

Theorem 1 *PTQU with fixed threshold p can be answered within $O(\log_B n + k/B)$ I/Os using $O(n)$ space when $p \geq 0.5$ and can be answered in $O(((1 -$*
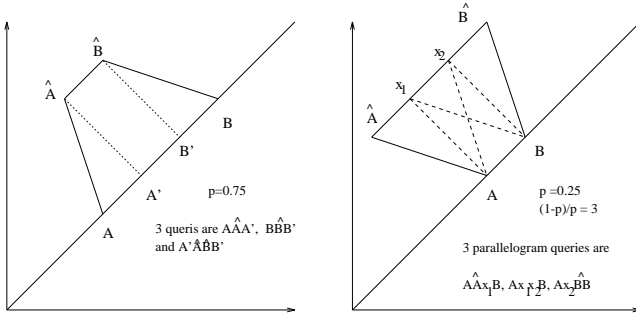
Figure 5: Probabilistic Threshold Queries with fixed Threshold

$p)/p)(\log_B n + k/B)$ *I/Os using* $O((1-p)n/p)$ *space when* $p < 0.5$.

Proof : Figure 5 shows the trapezoidal query regions for $p \geq 0.5$ and $p < 0.5$. When $p \geq 0.5$ the query can be answered by answering two wedge queries $A\hat{A}A'$, $B'\hat{B}B$ and a 3-sided parallelogram query $A'\hat{A}\hat{B}B'$. Recall that side AB is an open side since there are no points on the other side of AB. Hence, the open side for all these queries are along the side AB. Since the slopes of these lines only depend on p and p is fixed, all the slopes involved are fixed. Hence, the answer is the union of these 3 queries. Note that this requires maintaining three separate structures, one for each query.

When $p < 0.5$, the ratio of lengths of $\hat{A}\hat{B}$ and AB is $(1-p)/p$. The query can be answered using $(1-p)/p$ 3-sided parallelogram queries ($\hat{A}\hat{B}$ can be partitioned into $(1-p)/p$ segments which have the same length as AB and each segment forms a separate parallelogram query). The answer is the union of answers of these queries. This requires $(1-p)/p$ simultaneous structures and hence the space and time bounds. □

While these kind of bounds (provable data structures) do not exist when p is not fixed, we can still use 2D R-tree as in [13] to answer the trapezoidal queries. This forms the basis of our index in the next section.

5   An Extensive Uncertainty Index

In the previous section, we observed that if the uncertainty pdfs are uniform over each interval, then PTQ can be answered by indexing intervals as 2D objects and performing polygonal search queries over this index. However, this approach works only for uniform pdfs. When pdfs are allowed to be arbitrary, we cannot always obtain a boundary which separates intervals that satisfy the query from the intervals that do not. However, the pruning approach of Section 3 can be helpful. Here we explore how to combine both ideas to build an extensive index capable of handling arbitrary pdfs.

As discussed in the Section 3, a possible drawback of the simple R-tree based approach is that the MBR of

an R-tree can have a small interval as well as a large interval. Suppose a large MBR contains an interval much shorter than the length of the MBR, placed near the right boundary of the MBR. If the result of a PTQ contains the small interval, this implies the PTQ has to visit the large MBR just for the sake of the small interval. This can result in a larger search cost, particularly when the small interval in that MBR is all that the PTQ answer requires, but the large amount of space on the left of the small interval may have to be visited unnecessarily. Although, heuristics for R-tree (e.g., minArea) attempt to avoid such cases they cannot always acheive it.

For uniform pdfs, as we will see in the experiments, 2D indexing supporting trapezoidal queries shows improvement over PTI. Now consider the mapping of intervals to a 2D plane with interval $[x, y]$ mapped to the point (x, y). By rotating this mapping by 45 degrees clockwise and scaling it upwards by a factor of $\sqrt{2}$, the transformation represents the mean points of intervals along x axis, and the lengths of the intervals along the y axis. For uniform pdfs, the length of the interval is directly proportional to the standard deviation (which is one-third of the length). Thus, the 2D mapping when indexed using R-tree not only tends to put the intervals with close proximity (similar mean values) in the same MBR, but also ensures that all intervals in the same MBR have similar standard deviation values (or more generally, variances). We call this property of 2D indexing where intervals with similar variance values are grouped together as *variance-based clustering*. Let us have a closer look at how this property can be extended to handle general cases of pdfs.

Definition 5 *An x-deviation of a node N (either interval or an MBR) in a PTI is defined as* $(N.rb(x) - N.lb(x))/2$. *It is denoted by* $N.dev(x)$.

Note that $N.dev(x)$ decreases as x increases. Also, $N.dev(x)$ can be negative, if $x > 0.5$. In particular, when N is an interval (and not an MBR), $N.dev(x)$ is necessarily negative, if $x > 0.5$, in which case its left-x-bound and right-x-bound "swap" their positions.

Definition 6 *A set* T *of data items is variance-monotonic if for any two items* T_i *and* T_j, *their uncertainty pdfs* f_i *and* f_j *are such that for any values* $x, y \in (0, 1]$ *if* $|T_i.dev(x)| \geq |T_j.dev(x)|$ *then* $|T_i.dev(y)| \geq |T_j.dev(y)|$.

Definition 7 *A variance monotonic set* T *is variance-monotonic smooth if* $T_i.dev(x)/T_i.dev(y) = T_j.dev(x)/T_j.dev(y)$ *for any* i, j, x, y.

Definition 8 *A set* T *of data items is symmetric if its pdf is symmetric around the midpoint of the interval.*

Definition 9 *A set* T *of data items is regular if it is both symmetric and smooth variance-monotonic.*

Many standard pdfs like Gaussian[1] or uniform have the smooth variance-monotonic property. For example, consider two intervals $T_i = [9, 19]$ and $T_j = [35, 55]$ with uniform pdfs. Let us pick two arbitrary values for x and y, say 0.2 and 0.4. Then, $T_i.dev(x) = 6, T_i.dev(y) = 2, T_j.dev(x) = 12$ and $T_j.dev(y) = 4$. Thus, the ratio $T_i.dev(x)/T_i.dev(y) = T_i.dev(x)/T_i.dev(y) = 3$. Thus, for a smooth variance-monotonic set, this ratio only depends on x and y and is same for all the objects in the set when x and y are fixed. A set of data items which consists of all objects with Gaussian pdfs (each object can have different μ, σ) is smooth variance-monotonic. This is true also for a set of all the objects with uniform pdfs. However, if a set of objects has both kinds of pdfs uniform as well as Gaussian simultaneously, then it is no longer smooth variance-monotonic.

Given an interval with a pdf, we first determine its *representative deviation* $T_i.rdev$, defined in the following manner: Select some values $x_1, x_2, x_3, ..., x_j \in (0, 1]$. Then, for each i, $T_i.rdev$ is an aggregate function of $|T_i.dev(x_1)|, ..., |T_i.dev(x_j)|$. In this paper, we will simply take the aggregate function to be the average of these values. The values of $x_1, x_2, .., x_j$ are selected as some of the most relevant thresholds for the index. Note that for smooth variance monotonic data set just one value of x can give the representative deviation. We also calculate an entity called $T_i.mean$ which is the point in the interval such that there is 50% probability on either side of the point (i.e. mean value of the interval according to the pdf).

5.1 Uncertainty Indexing for Regular Sets

As noted earlier, both Gaussian and uniform pdfs are symmetric and smooth variance-monotonic. Thus a set consisting of all Gaussian pdfs is a regular set. For indexing a regular set, we can use a 2D R-tree. First a representative threshold value x is selected (say 30%). Then we calculate $T_i.mean$ and $T_i.rdev$ for all items in T. For different values of $y \in (0, 1]$, a table of ratios of $r(y) = T_i.dev(y)/T_i.dev(x)$ is also calculated. This table is called the *ratio table* and is kept in the main memory. Note that this ratio is the same for each object in T. We index each item by its 2D coordinates $(T_i.mean, T_i.rdev)$, and construct a 2D R-tree on this representation.

To process a query (a, b, p), we first check the query against MBRs of the nodes in this tree. A node N is pruned when it is guaranteed that no item in the subtree rooted at N can satisfy (a, b, p). Let μ_1, μ_2 be the lowest and highest values of $T_i.mean$ over all objects in the subtree of N and let σ_1, σ_2 be the lowest

and highest values of $T_i.rdev$ over all the objects in the subtree. Note that the MBR for N is $[(\mu_1, \sigma_1) : (\mu_2, \sigma_2)]$.[2] Let $\hat{p} \leq p$ be the value where the ratio $r(\hat{p})$ is pre-calculated in the ratio table. We calculate two values L, R such that $L \leq N.lb(\hat{p}) \leq N.lb(p)$ and $R \geq N.rb(\hat{p}) \geq N.rb(p)$. Note that L may be greater than R, when $\hat{p} \geq 0.5$. If $\hat{p} < 0.5$ then, $L = \mu_1 - r(\hat{p})\sigma_2$ and $R = \mu_2 + r(\hat{p})\sigma_2$. If $\hat{p} \geq 0.5$ then, $L = \mu_1 + r(\hat{p})\sigma_1$ and $R = \mu_2 - r(\hat{p}\sigma_1$. In the case where $L \leq R$, if the range of the query $[a, b]$ does not overlap with $[L, R]$ then we can safely prune N. If $L > R$ then the range of the query $[a, b]$ must contain $[R, L]$ to not prune N. If it does not contain, then N is pruned. The following theorem proves the correctness of the method.

Theorem 2 *If a node N with MBR $[(\mu_1, \sigma_1) : (\mu_2, \sigma_2)]$ is pruned by the query (a, b, p) where the set of data items T is regular, then there is no data item in the subtree of N which satisfies the query (a, b, p).*

Proof : Consider the case when $p < 0.5$ and N is pruned. Here, $L \leq R$ and $[a, b]$ does not overlap with $[L, R]$. Without loss of generality, we assume that $b < L$. We prove our claim by contradiction. Assume that there is an object T_i in the subtree of N satisfying the query. Now, $T_i.mean \geq \mu_1$ and $T_i.rdev \leq \sigma_2$. Let $L' = T_i.mean - T_i.dev(\hat{p})$. Thus, by symmetry and the definition of $T_i.dev$, $L' = T_i.lb(\hat{p})$. Since $r(\hat{p}) = T_i.dev(\hat{p})/T_i.rdev$, $L' \geq L$. Also, $\hat{p} < p$ implies T_i satisfies the query $(a, b, \hat{p})$, This means $L' \leq b$ implies $L \leq b$, which is a contradiction. Hence, such an object T_i cannot exist. The proofs for all other cases are exactly the same with appropriate parameters changed, which we skip due to limitation of space. ∎

Compared with PTI, this scheme has an advantage in terms of space. Recall that PTI requires extra space to store probability threshold information. With this scheme, we exploit the fact that the data set is smooth variance-monotonic and symmetric, and compute the probability threshold information "on the fly". Thus overhead required by PTI can be avoided.

5.2 Uncertainty Indexing for Arbitrary pdfs

For arbitrary pdfs (not necessarily variance monotonic), the correctness of the above approach cannot be guaranteed. We need to revert to the PTI structure of Section 3. Hence, apart from the MBR boundaries we also maintain the boundaries from certain fixed values of probability threshold. We can build an index based on 2D R-tree, with operations like insert, delete and split. For this we index each item by $(T_i.mean, T_i.rdev)$. Here, $T_i.rdev$ is calculated as an average of $T_i.dev(x)$ for some predetermined values of x.

---

[1]When we assume Gaussian pdf the length of the interval may be considered as infinite. Alternatively we may use an approximation of Gaussian distribution by trimming away the portion of the interval beyond which the probability is below a certain threshold and normalizing the pdf inside the interval

[2]We use $(lx, ly) : (rx, ry)$ to represent a rectangle where (lx, ly) and (rx, ry) are the coordinates of the lower and upper bounds respectively.

However, this structure does not guarantee query correctness and does not allow pruning if we do not include PTI information. Hence, for query processing, we consider each node N's MBR as a one-dimensional object. In PTI structure, we first include $N.lb(0)$ and $N.rb(0)$. This forms the MBR of T_j when it is considered as a one-dimensional entity. Then we also include $N.lb(x)$ and $N.rb(x)$ for various predetermined values of x depending upon likely query thresholds. Then query pruning is done based on this PTI structure given in Section 3. Thus this structure is constructed as a 2D R-tree, but its query processing is treated as a 1D R-tree operation. The main difference between this structure and the one in Section 3 is that this structure attempts directly to cluster the data items with similar variance values. We will see in Section 6 that the variance-clustered R-tree improves the query performance.

6 Experimental Results

In this section we present the performance results of an extensive simulation for the index structures we proposed. We implemented a one-dimensional R-tree without uncertain information augmented (hereby referred to as *R-tree*), a PTI, as well as the extensive uncertainty scheme (referred to as *extensive*), where both the ideas of PTI and two-dimensional variance-based-clustering techniques applied. We will discuss our simulation model followed by experimental results.

6.1 Simulation Model

We generated two sets of data. The first set of data are uncertain data, with their lengths uniformly distributed in $[U_{min}, U_{max}]$, with a uniform uncertainty pdf. The second set of data are the properties of probabilistic threshold queries. Similar to the uncertain data, the length of each query range is also normally distributed with U_{mean} and U_{dev}, with a uniform uncertainty pdf. Their probability thresholds are uniformly distributed between 0.1 and 1. Table 1 presents the parameters for uncertain data and PTQ.

The total insertion and query I/O performance of the indexes is measured. Each disk page contains S_{page} bytes, with N_{entry} entries per page. By default, a PTI node contains five tuples of x-bounds, where $x \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

6.2 Results

6.2.1 Scalability of Uncertainty Indexes

In the first experiment, we examine the scalability of the three indexes. Figure 6 shows their I/O performance as the number of items are increased from 25K to 100K. We can see that the total number of I/Os for updates and queries for all three indexes increase linearly with the data set size. This is because all the lengths of queries are normally distributed with μ 100

| Param	Default	Meaning
Uncertain Data		
N_{int}	$100K$	# of intervals
L_{min}	0	Min value of L_i
R_{max}	$10,000$	Max value of U_i
U_{min}	10	Min value of $U_i - L_i$
U_{max}	$1,000$	Max value of $U_i - L_i$
$f_i(x)$	$1/(U_i - L_i)$	Uncertainty pdf
Probabilistic Threshold Queries		
N_{int}	$10K$	# of range queries
a_{min}	0	Min of lower bound(a)
b_{max}	$10,000$	Max of upper bound(b)
I_{mean}	100	Mean of interval length
I_{dev}	10	Deviation of interval length
p	$[0.1, 1]$	Prob. threshold (uniform)
Tree parameters		
S_{page}	4096	Size of a page (bytes)
N_{entry}	20	# of entries (per page)

Table 1: Parameters and baseline values.
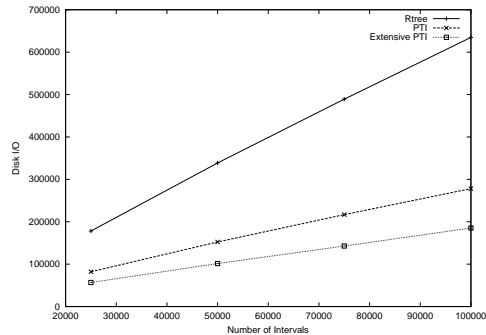
Figure 6: Total I/O vs. Number of Intervals

and σ 10 throughout the experiment while the data size increases. As a result, the number of items that satisfy the queries increases linearly with number of items.

A more interesting observation is that both PTI and *extensive* perform much better than the R-tree. Indeed, PTI almost spent about 50% of the I/Os required for R-tree, while *extensive* needs only about 30% of R-tree's effort. The reason is simply because R-tree does not use uncertainty pdf while the index is being accessed. As a result, many items that do not satisfy the probability thresholds of queries are not pruned. This results in a large set of intervals being needed to be examined. On the other hand, both PTI and *extensive* are designed to be sensitive to uncertainty pdf information during the searching-and-pruning process. Thus, they are able to prune much more items away than R-tree. Notice that although here we only show the number of I/Os, answering PTQ with an R-tree may even suffer more in terms of computation time, because more items are obtained after the index is searched, and the time required for the integration op-
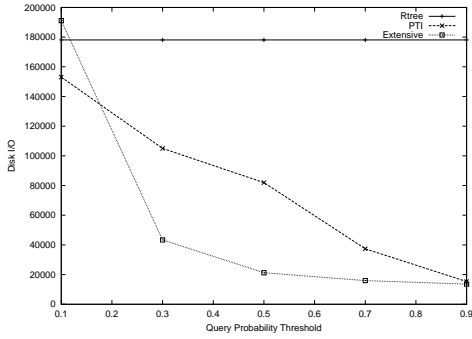
Figure 7: Total I/Os vs. Query Probability Threshold

eration to evaluate the probability will be significantly increased.

We can also see that *extensive* requires about 50% less I/Os than PTI. This is because *extensive* applies the variance-based clustering techniques, while PTI does not. In a PTI node, it is possible for a small interval to be stored in a relatively large MBR. Thus, if that small interval satisfies a PTQ (which is easy because of its small size), the large MBR has to be visited, and the query may need to spend time on the space in the large MBR, which can be avoided in the *extensive* scheme.

6.2.2 Effect of Query Probability Threshold

In the second experiment, we study the effect of probability thresholds (p) of queries on I/O performance. To obtain a data point, the threshold value of each query is made the same in (0,1). The results are shown in Figure 7, which illustrates the three indexes' performance under different values of p. We can observe that as p increases, the number of I/Os requried for both PTI and *extensive* decreases. This is simply due to the reduction in the number of qualified intervals for the queries with more stringent probability threshold requirements. Nonetheless, R-tree does not take advantage of this at all, since it only prunes away items that do not overlap the queries, regardless of their threshold requirements.

The performance of PTI improves as p increases. Consider a query with some amount of overlap with an MBR. When its threshold increases, it has more chance to use an x-bound. For *extensive*, the size of the trapezoidal region is inversely proportional to p and hence its performance also improves. The advantages of *extensive* over PTI are more pronounced when p is betwen 0.3 and 0.7. When p is 0.5, *extensive* requires four times fewer I/Os than PTI. At the extreme values of p, the advantages of *extensive* are not as pronounced, becasue the query either becomes a containment query or an overlap query. Hence, both PTI and *extensive* perform similarly.

7   Related Work

The probabilistic uncertainty model for sensor data studied here is a modified version of the one discussed in [9]. While we assume the bounds of uncertain intervals are constant, in that paper the uncertain intervals are time-varying functions. That paper also presents a taxonomy of different representations of data uncertainty in terms of intervals. The probabilistic uncertainty model for two-dimensional moving objects is discussed in [8]. Yazici et al. [21] discusses uncertainty in different data types, such as sets, intervals and fuzzy data.

In [7], a general classification, evaluation and quality of different types of probabilistic queries for sensor data are presented. Probabilistic queries in moving object databases are studied in [20] and [8], where range query and nearest-neighbor query algorithms respectively are presented. The probabilistic threshold query for sensor data is proposed in [9], where efficient computation strategies of probability values by exploiting probability thresholds are discussed. However, it does not address indexing of imprecise data.

There are numerous works in the field of interval indexing. In [14][3], the idea of mapping intervals as points in two-dimensional space is discussed. They also talk about the transform of one-dimensional stabbing queries and range queries to two-sided orthogonal queries in two-dimensional space. Manolopoulos et al. [17] propose an efficient interval tree to facilitate the execution of intersection queries over intervals. Kriegel et al. [15] discusses an implementation of interval trees, which is conveniently built on top of relational tables, and algorithms are expressed as SQL queries.

Different types of range queries in two dimensional space have been well studied. For half-space queries, lower bounds are discussed in [12][4], and optimal data structures are presented in [1][10]. The lower bounds of simplex queries are derived by Chazelle [6]. Goldstein et al. [13] modifies the R-tree to answer simplex queries, which works well in practice. Optimal data structures for answering 2-sided and 3-sided queries, which have fixed slopes but not necessarily orthogonal, are discussed in [2]. A comprehensive survey on geometric range searching can be found in [18][11].

Although a rich vein of work exists in interval indexing, the issue of indexing uncertain data that involves probability computation has not been well addressed. A recent paper by Lin et al. [16] discusses an extension of the TPR-tree [19] to index trajectories of moving objects, where each point in the trajectory has a rectangular uncertain bound. We study the indexing of general sensor data, and establish a theoretical foundation of the problem. We also propose novel indexing techniques for fixed/variable probability thresholds, and different kinds of pdfs. To our best knowledge, these questions have not been answered previously.

8   Conclusions

Uncertainty is an important emerging topic in sensor databases. In this paper we investigated the problem of indexing uncertain data. We showed that this problem is theoretically difficult, by showing how it can be transformed to a polygonal range query in two-dimensional space. However, heuristics do exist to handle the problem efficiently in practice. In particular, we proposed the ideas of augmenting probability threshold bounds to an index, as well as clustering intervals based on their statistical information. Our extensive experiments showed that these two ideas, when used together, can significantly improve the performance of probabilistic threshold queries.

There are numerous avenues for future work. We will study the uncertainty indexing problem for other kinds of queries. The indexing of other types of uncertain data, such as sets and fuzzy data, are also worth investigating. We are also interested in the issue of efficient indexing of uncertain intervals with time-dependent bounds.

References

[1] Pankaj K. Agarwal, David Eppstein, and Jirí Matousek. Dynamic half-space reporting, geometric optimization, and minimum spanning trees. In *FOCS*, pages 80–89, 1992.

[2] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *PODS*, pages 346–357, 1999.

[3] L. Arge and J. S. Vitter. Optimal dynamic interval management in external memory (extended abstract). In *FOCS*, pages 560–569, 1996.

[4] H. Brönnimann, B. Chazelle, and J. Pach. How hard is half-space range searching. *Discrete and Computational Geometry*, 10:143–155, 1993.

[5] C. Chatfield. *The analysis of time series an introduction*. Chapman and Hall, 1989.

[6] Bernard Chazelle. Lower bounds on the complexity of polytope range searching. *Journal of American Mathematical Society*, 2:637–666, 1989.

[7] 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.

[8] R. Cheng, D. V. Kalashnikov, and S. Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Knowledge and Data Engineering (To appear)*, 2004.

[9] R. Cheng and S. Prabhakar. Managing uncertainty in sensor databases. In *SIGMOD Record issue on Sensor Technology*, December 2003.

[10] Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 2:195–222, 1987.

[11] Jeff Erickson and Pankaj K. Agarwal. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, Contemporary Mathematics 223:1–56, 1999.

[12] Michael L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM J. Comput.*, 10(1):1–10, 1981.

[13] Jonathan Goldstein, Raghu Ramakrishnan, Uri Shaft, and Jie-Bing Yu. Processing queries by linear constraints. In *PODS*, pages 257–267, 1997.

[14] Paris C. Kanellakis, Sridhar Ramaswamy, Darren Erik Vengroff, and Jeffrey Scott Vitter. Indexing for data models with constraints and classes. *J. Comput. Syst. Sci*, 52(3):589–612, 1996.

[15] H. Kriegel, M. Potke, and T. Seidl. Managing intervals efficiently in object-relational databases. In *Proc. of the 26th Intl. Conf. on VLDB*, Cairo, Egypt, 2000.

[16] B. Lin, H. Mokhtar, R. Pelaez-Aguilera, and J. Su. Querying moving objects with uncertainty. In *Proceedings of IEEE Semiannual Vehicular Technology Conference*, 2003.

[17] Y. Manolopoulos, Y. Theodoridis, and V.J. Tsotras. Chapter 4: Access methods for intervals. In *Advanced Database Indexing*. Kluwer, 2000.

[18] Jirí Matousek. Geometric range searching. *ACM Comput. Surv.*, 26(4):421–461, 1994.

[19] S. Saltenis, C. Jensen, S. Leutenegger, and M. Lopez. Indexing the position of continuously moving objects. *Proc. of ACM SIGMOD*, 2000.

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

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