Efficient Range Query Processing on Uncertain Data

Andrew Knight
Rochester Institute of Technology
Department of
Computer Science
Rochester, New York, USA
andyknig@gmail.com

Qi Yu
Rochester Institute of Technology
Department of
Information Sciences and Technologies
Rochester, New York, USA
qyu@it.rit.edu

Manjeet Rege
Rochester Institute of Technology
Department of
Computer Science
Rochester, New York, USA
mr@cs.rit.edu

Abstract

Uncertain data has emerged as a key data type in many applications. New and efficient query processing techniques need to be developed due to the inherent complexity of this new type of data. In this paper, we investigate a special type of query, range queries, on uncertain data. We propose a threshold interval indexing structure that aims to balance different time consuming factors to achieve an optimal overall query performance. Experimental results are presented to justify the efficiency of the proposed query processing technique.

1 Introduction

Uncertain data has been increasingly generated from a great variety of applications, such as scientific measurements, sensor networks, GPS, and mobile object tracking. In contrast to certain data whose values are exact constants, uncertain data take values that are described by probability measures, most notably probability distribution functions (PDFs). Due to the inherent complexity of uncertain data, new techniques need to be developed in order to efficiently process queries against this new type of data. A set of novel indexing structures have been developed to accelerate query processing. Representative ones include threshold index [3, 4], the U-tree [5], and 2D mapping techniques [1, 3]. Most of these approaches assume that disk I/Os are the dominating factor that determines the overall query performance. Thus, the indexing structures are usually designed to optimize the number of disk I/Os. However, uncertain data is inherently more complicated than certain data. Computing a range query on uncertain data usually involves complicated computations, which incur high CPU cost. This makes disk I/Os no longer the solely dominating factor that determines the overall query performance. Therefore, new indexing strategies need to be developed to optimize the overall performance of range queries on uncertain data.

Uncertain continuous data is usually modeled by PDFs. Some PDFs, like the uniform PDF, may be very simple to calculate. However, many widely used PDFs involve complicated computations, such as multimodal probability models for cluster analysis [6]. Computing a complicated PDF may involve high computational cost. Numerical approaches, such as Monte Carlo integration, have been exploited to improve the performance [5]. Riemann sum, however, provides a better strategy for one-dimensional cases. Even though Riemann sums can be faster than Monte Carlo integrations, they still incur high computational cost. Especially when a high computation accuracy is required, probability calculations may take even longer than disk I/Os. Therefore, the number of probability calculations must also be considered if the distribution of the uncertain data is complicated. In this case, the indexing strategy needs to balance between disk I/Os and the CPU cost to achieve an optimal overall query performance.

In this paper, we present a novel indexing strategy focusing on one-dimensional uncertain continuous data, called

IEEE IRI 2011, August 3-5, 2011, Las Vegas, Nevada, USA
978-1-4577-0966-1/11/$26.00 ©2011 IEEE

threshold interval indexing. It addresses the limitations of existing indexing structures on uncertain data, particularly for handling complicated PDFs, by treating uncertain objects as intervals and thereby leveraging interval tree techniques. The proposed indexing structure is also inspired by the optimized interval techniques from [2] to build a dynamic primary tree and store objects in nodes at different levels depending on the objects' sizes. The notion of using an interval tree to index uncertain data was suggested by Cheng et al. in [3] but disregarded in favor of an R-tree with extra probability limits called x-bounds. We assert that x-bounds can just as easily be applied to interval trees to index uncertain data with special benefits.

The rest of the paper is organized as follows. Section 2 gives the problem statement and provides an overview of previous research. Section 3 presents the threshold interval index. Section 4 gives experimental results of our two indexes versus the probability threshold index. Section 5 concludes the paper by offering direction for future research.

2 Problem Statement and Related Work

In this section, we start by providing a formal problem statement of range query on uncertain data. We then give an overview of related works in this area.

2.1 Problem Statement

Given a database table T, a *query interval* $[a, b]$ for an uncertain attribute e of an object u_i, and a *threshold probability* τ, a *range query* returns all uncertain objects u_i from T for which $Pr(u_i.e \in [a, b]) \geq \tau$.

Theoretically, an uncertain object could have more than one uncertain attribute. However, for this paper, we focus on indexing objects based on only one uncertain attribute.

2.2 External Interval Tree Index

Arge et al. [2] propose two optimal external interval tree indexes. Interval trees are not specifically designed for handling uncertain data, but one-dimensional uncertain objects may be treated as intervals by using their PDF endpoints. Both indexes use a primary tree for layout and secondary structures to store the objects at each node, but one has a dynamic primary tree instead of a static one. *Stabbing queries* are used to return results. However, the downfall of both interval indexes is that if many uncertainty intervals overlap with the query interval's endpoints, then few objects are pruned from the search, and a lot of time is wasted in calculating probabilities.

2.3 Probability Threshold Index

The *probability threshold index* (PTI) [3] allows range queries to prune more branches from searching than interval indexes allow by using a one-dimensional R-tree as a base tree with stricter boundaries called *x-bounds*. The PTI has many advantages. It is an elegant solution, and it is fairly easy to implement. The tree is dynamic as well. All boundaries are calculated when objects are added. Multiple x-bounds can be stored in each node, so queries can choose the most appropriate bounds for its threshold. Required storage space for internal nodes is relatively small.

The PTI is not without weaknesses, however. The primary weakness pointed out by Cheng et al. is that differences in interval sizes will skew the balance of the tree [3]. Cheng et al. also do not provide an optimal rectangle layout strategy for the PTI's base tree, the R-tree. The best strategy for any R-tree is to make MBRs as disjoint as possible. When MBRs overlap too much, extra disk I/Os and probability calculations must be performed because fewer nodes can be pruned. Adding new objects, especially objects of vastly different interval lengths, exacerbate overlap. Simply put, sloppy R-trees are inefficient, but optimal R-trees are very difficult to maintain.

When rectangles overlap, not all objects which fall completely within the query interval can be immediately accepted. Since MBRs might overlap, every node must be checked. There is no exclusivity between node intervals. Nodes may not be stored in any order if their intervals are stretched. Objects might appear in the overlapping portions of nodes, too. These compounding factors force probability calculations on all objects in each unpruned node. This wastes lots of time, especially when the query interval is much larger in size than most uncertainty intervals.

2.4 2D Mapping Indexes

Cheng et al. first suggested 2D mapping techniques as an alternative to the PTI for uniform PDFs [3]. Agarwal et al. then expanded 2D mapping techniques to histogram PDFs [1]. Histogram PDFs can easily be transformed into linear piecewise threshold functions and stored as a set of line segments. The structures of the indexes presented in [1] manipulate the line segments. They are efficient for uniform and histogram PDFs, but they are inapplicable for more general PDFs. Furthermore, each index is rigidly based upon one threshold value; separate indexes must be constructed for additional thresholds. This is starkly different from the PTI, which can manage several threshold values in one structure.

3 Processing Range Queries on Uncertain Data

We now present the proposed range query process technique for uncertain data. The cornerstone is the *threshold interval index* (TII). The TII is in essence a combination of a dynamic external interval tree and the x-bounds structure used by PTI. This structure presents two key advantages. The first advantage is that the structure intrinsically and dynamically maintains balance all the time. The second advantage is that the interval-based structure makes all uncertain objects which fall entirely within the query interval easy to find and, therefore, possible to add to the results set without further calculation. The PTI does not allow this because its MBRs might overlap. Furthermore, adding x-bound avoids the interval index's problem for when many uncertainty intervals overlap the query interval.

3.1 TII Structure

The TII has a primary tree to manage interval endpoints. It also has secondary structures at internal nodes of the primary tree to store objects. When an object is added to the index, the endpoints of its uncertainty interval are added to the primary tree. Then, the object itself is added to the secondary structures of the appropriate tree node. Each object is also assigned a unique id if it does not already have one. X-bounds are stored for each internal node.

3.1.1 Primary Tree

The primary tree is a *weight-balanced B-tree* with branching parameter $r > 4$ and leaf parameter $k > 0$. The *weight* of a node is the number of items (in this case, endpoints) below it. All leaves are on level 0. All endpoints are stored at the leaves, and internal nodes hold copied values of endpoints. The weight-balanced B-tree provides an effective way to dynamically manage intervals and spread. Arge et. al. describe this tree in detail, including time bounds, in [2].

3.1.2 Secondary Structures

Each internal node v represents an interval I_v, which spans all interval endpoints represented by children of v. Thus, the c children of v (for $\frac{1}{4}r \leq c \leq 4r$) naturally partition I_v into subintervals called *slabs* [2]. Each slab is denoted by I_{v_i} (for $1 \leq i \leq c$), and a contiguous region of slabs, such as $I_{v_2}I_{v_3}I_{v_4}$, is called a *multislab* [2]. All slab boundaries within I_v are stored in v. Note that I_{v_i} is the interval for the child node v_i.

An uncertain object is stored at v if its uncertainty interval falls entirely within I_v but overlaps one or more boundaries of any child node's I_{v_i}. (A leaf stores uncertain objects whose PDF endpoints are contained completely within the leaf's interval endpoints.) Each object is stored at exactly one node in the tree, as shown in Figure 1. Let U_v denote the set of uncertain objects stored in v. In the external dynamic interval index, these objects are stored in secondary structures called *slab lists* [2], partitioned by the slab boundaries. However, only two secondary structures are needed per node for the TII because range queries (described later in this section) work slightly differently than stabbing queries. The *left endpoint list* stores all uncertain objects in increasing order of their uncertainty intervals' left endpoints. The *right endpoint list* stores all uncertain objects in increasing order of their uncertainty intervals' right endpoints. This is drastically simpler than the optimal external interval tree, which requires a secondary structure for each multislab [2]. If the uncertain objects hold extra data or large PDFs, it might be advantageous to store only uncertainty interval boundary points and object references in the two lists. The actual objects can be stored in a third structure to avoid duplication.

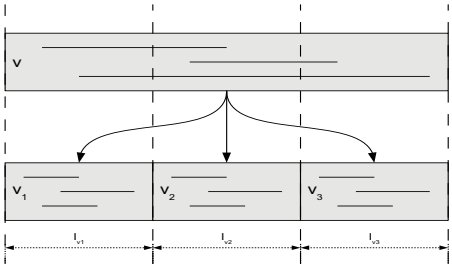


Figure 1: A node v with three child nodes. The dotted lines denote slab boundaries. Note how objects are only stored within intervals which can completely contain them.

3.1.3 Applying X-bounds

X-bounds were introduced as part of the probability threshold index [3] and can easily be applied to the TII.

Definition 1 *An* x-bound *is a pair of values* (L_x, R_x) *for a continuous PDF* $f(s)$ *with uncertainty domain* $[L, R]$ *such that*

$$x = \int_{L}^{L_x} f(s)ds = \int_{R_x}^{R} f(s)ds \tag{1}$$

L_x is the *left x-bound*, and R_x is the *right x-bound*. Since the domain of $f(s)$ is $[L, R]$, L_x and R_x are unique. Note that x is a probability value, meaning $0 \leq x \leq 1$. For example, if $x = 0.25$, then there is a 25% chance that the object's value appears in the interval $[L, L_{0.25}]$. Furthermore, there would be a 25% chance it appears in $[R_{0.25}, R]$ and a 50% chance it appears in $[L_{0.25}, R_{0.25}]$. Note also that if $x > 0.5$, then $R_x < L_x$.

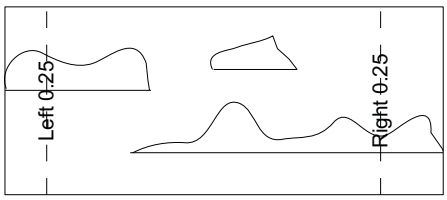


Figure 2: A node with x-bounds and objects. X-bounds are calculated for each node based on uncertain objects' PDFs. The left and right 0.25-bounds are tighter than the MBR.

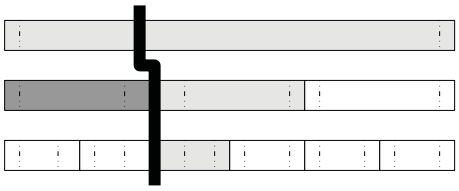


Figure 3: A left stab is denoted by the thick black line. The light gray nodes are visited by the stab. The white nodes are not visited. The dark gray node is pruned based on x-bounds.

The notion of x-bounds can be applied to tree nodes as well as to PDFs, as seen in Figure 2. The left x-bound for a node is the minimum left x-bound of all child nodes and objects, and the right x-bound is the maximum right x-bound of all child nodes and objects. Specifically, for a node v, left and right x-bounds are calculated for I_v. A child node's x-bounds must be considered when calculating v's x-bounds: a child node might have tighter x-bounds than any of the uncertain objects stored at v. The interval I_v accounts for all uncertain objects stored at v and in any child nodes of v, and so should the x-bounds. The x-bounds for v's slabs are given by the x-bounds on v's child nodes. All of v's x-bounds are stored in v's parent. In this way, the interval I_v is analogous to a minimum bounding rectangle in an R-tree, and intervals are tightened by x-bounds in the same way as MBRs are tightened in the PTI [3]. X-bounds for more than one probability x can be stored as well.

3.2 Range Query Evaluation

Evaluating range queries for objects in $[a, b]$ with a threshold τ on the TII is like evaluating stabbing queries on a regular interval tree. Two *stabs* are executed for each endpoint of the query interval: a left stab and a right stab. The nature of the query forces these stabs to be performed slightly differently from how they are described in [2]. Once the stabs are made, a series of *grabs* can be performed for all objects in between. This is called the *stab 'n grab search*.

3.2.1 The Left Stab

The *left stab* is the most complicated part of the stab 'n grab search. The search starts at the root node and continues down one path through child nodes until it hits the leaf containing the closest x-bound to a within its boundaries. This leaf is called the *left boundary leaf*. X-bounds are used to prune this search. If a node's right x-bound is less than a or if a node's left x-bound is greater than b, then the node can be pruned, because the probability that any of its objects falls within the query interval must be less than the query's threshold. Objects are checked at nodes along the stab to see if they belong to the result set.

Before moving to the next child node, the uncertain objects stored in secondary structures at the current node must be investigated, because their uncertainty intervals may overlap the query interval. If they overlap the query interval, then they might be valid query results. Between the secondary structures, only the right endpoint list is needed. A quick binary search can be performed to find which objects fall within the query interval. Any object whose right endpoint is less than a can be disregarded. Any object whose both endpoints are within the query interval is added to the result set automatically. Otherwise, a probability calculation must be performed using the object's PDF to determine if it meets the threshold probability. The same strategy applies for the left boundary leaf. All valid objects are added to the result set.

3.2.2 The Right Stab

The *right stab* is analogous to the left stab, except it searches with b instead of a. The leaf found at the bottom of the stab is called the *right boundary leaf*. X-bound pruning is performed for the rightmost child nodes, not the leftmost. The process for searching the secondary structures is the same as in the left stab, except "left" and "right" are switched wherever mentioned. Furthermore, nodes visited during the left stab can be skipped during the right stab, because the process for investigating uncertain objects accounts for both endpoints of the uncertainty interval. This is why references to visited nodes are stored during the left stab.

3.2.3 The Grabs

The two stabs find the two boundary leaves and some uncertain objects in the result set. The remaining objects to investigate reside in the nodes between the two boundary leaves. Thankfully, all objects in between can be added to the result set without any probability calculations. Remember, intervals for nodes on the same level do not overlap, so all

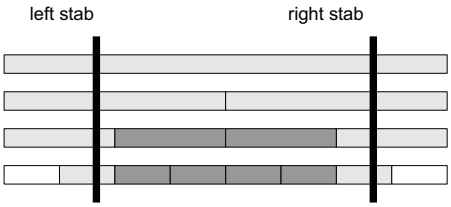


Figure 4: A stab 'n grab query. The light gray nodes are visited during the stabs, and the dark gray nodes are visited during the grabs. Note how grabbed nodes fall completely within the query interval.

objects stored at nodes between the boundary leaves must fall entirely within the query. The most effective way to grab all of these uncertain objects is to perform a post-order tree traversal starting at the left boundary leaf and ending at the right boundary leaf, skipping each node that has already been visited. No extra searching needs to be done on the secondary structures. Figure 4 illustrates a full stab 'n grab query.

3.2.4 Time Bounds

A range query can be answered within the following time bounds using the stab 'n grab search:

Theorem 1 *Let I be a TII storing N uncertain objects, whose primary tree has branching parameter r and leaf parameter k. Assume any calculation on an uncertain object's PDF takes $O(d)$ time. A range query Q with query interval $[a, b]$ and threshold τ can return all T uncertain objects stored in I which fall within the query interval with probability $p \geq \tau$ in $O(kd\, log_r(N/k) + T/k)$ time.*

4 Experimental Results

This section presents an evaluation of our experimental results. We use the probability threshold index as a benchmark against which to test the proposed threshold interval index. The PTI used is a "practical" PTI in that objects were removed and replaced to introduce a small level of skew.

The purpose for testing these twp indexes is to compare their range query performance. Performance is measured by three primary metrics: number of disk I/Os, number of probability calculations, and runtime (in milliseconds).

Four different performance tests are run. Each test builds the indexes from a common data set and runs range queries on each index. Descriptions are given in Table 1. Datasets are generated synthetically. Uncertain objects contain two attributes: an id and a PDF. The PDF interval is determined randomly based on test parameters, given in Table 2. X-bounds are calculated for the probability values $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ on each index. The block size is 4096 bytes.

Table 1: Performance Tests

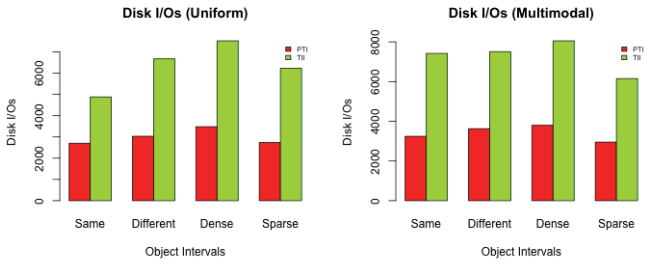
Test	Description
`Same`	object uncertainty intervals have the same length
`Different`	object uncertainty intervals have different length
`Dense`	many objects overlap
`Sparse`	objects are spaced out

Table 2: Test Parameters

Parameter	Same	Different	Dense	Sparse
Num Objects	10000	10000	10000	10000
Min Object Value	0	0	0	0
Max Object Value	10000	10000	1000	1000000
Min PDF Length	100	50	1	1
Max PDF Length	100	500	100	10

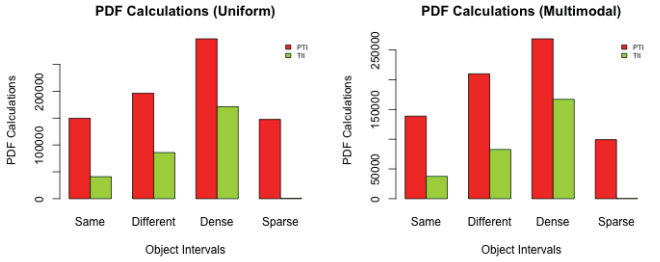
Each test is run with two types of PDFs. Just like for previous tests against the PTI, one PDF used is a uniform PDF [3]. The second PDF is the multimodal Gaussian distribution with four peaks, which is significantly more complicated. Each PDF can be stored by left and right endpoints and can be stretched to the appropriate interval length. All probability calculations are performed by using Riemann sums. 1000 rectangles are used for each Riemann sum to keep the average error margin around 0.1%. Range queries must also be generated. For each test, 100 queries are generated. Each query interval is random within a given domain. Each query is run against each index, and results for all queries are tabulated aggregately. The probability threshold used is $\tau = 0.3$.

The `Same`, `Different`, `Dense`, and `Sparse` tests all test the spread of uncertain objects. Figures 5 gives results for these tests. It is clear that object spread and size significantly affects performance. All indexes have worse performance for objects of different lengths and for densely clustered objects. What is interesting is the difference in performance metrics. The PTI uses fewer disk I/Os than the TII. Although for objects of the same size the PTI is comparable for uniform PDFs, the TII generally use about 1.5 to 2 times as many disk I/Os. The PTI is far surpassed, however, in regards to the number of probability calculations. Threshold indexes typically use only half to a third of the number of calculations as the PTI. This number is most staggering for sparse indexes: the TII makes relatively no calculations. This is expected due to the extra pruning done



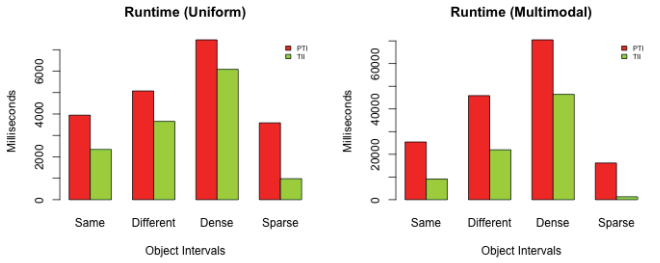
(a) Disk I/Os for Uniform PDFs

(b) Disk I/Os for Multimodal PDFs



(c) Uniform PDF Calculations

(d) Multimodal PDF Calculations



(e) Runtime for Uniform PDFs

(f) Runtime for Multimodal PDFs

Figure 5: Performance results for `Same`, `Different`, `Dense`, and `Sparse` tests.

by the TII's stab 'n grab query. Overall, the total runtime favors threshold indexes for complicated probability functions.

The trends between uniform and multimodal PDFs are generally the same for disk I/Os and probability calculations. This is not too surprising, because PDF shape has only a small affect on index structure. The major difference is in total runtime, as seen in Figures 5e and 5f. Since the multimodal PDF is more complicated, calculations take longer. Thus, the margin by which the TII outperforms the PTI is much larger for multimodal PDFs than for uniform PDFs.

5 Conclusion

In this paper, we present threshold interval indexing, a new strategy for indexing complicated uncertain continuous data of one dimension. The key advantage of threshold interval indexing over existing strategies, such as the probability threshold index, is that it handles complicated PDFs much more efficiently because it handles balance better with its intervals. Although our paper focuses on range queries, the indexing structure could also be used for other types of queries, such as joins. It is also important for uncertain data strategies to be incorporated into database management systems. Parallelization should also be explored further.

References

[1] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *Symposium on Principles of Database Systems (PODS)*, pages 137–146, 2009.

[2] L. Arge and J. S. Vitter. Optimal external memory interval management. *SIAM Journal on Computing*, 32(6):1488–1508, 2003.

[3] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proc. International Conference on Very Large Data Bases (VLDB)*, pages 876–887, 2004.

[4] S. Prabhakar, R. Shah, and S. Singh. *Managing and Mining Uncertain Data*, chapter Indexing Uncertain Data, pages 299–325. Springer, 2009.

[5] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *Proc. International Conference on Very Large Data Bases (VLDB)*, 2005.

[6] J. Yu, M.-S. Yang, and P. Hao. A novel multimodal probability model for cluster analysis. In *RSKT '09: Proceedings of the 4th International Conference on Rough Sets and Knowledge Technology*, pages 397–404, Berlin, Heidelberg, 2009. Springer-Verlag.