Probabilistic threshold range aggregate query processing over uncertain data

Author:
Yang, Shuxiang

Publication Date:
2009

DOI:
https://doi.org/10.26190/unsworks/19575

License:
https://creativecommons.org/licenses/by-nc-nd/3.0/au/
Link to license to see what you are allowed to do with this resource.

Downloaded from http://hdl.handle.net/1959.4/43374 in https://unsworks.unsw.edu.au on 2024-05-01

Probabilistic Threshold Range Aggregate Query Processing over Uncertain Data

BY

Shuxiang Yang

B.Sc., Wuhan University, 2004

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

IN THE SCHOOL

OF

Computer Science and Engineering

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

February 21, 2009

All rights reserved. This work may not be

reproduced in whole or in part, by photocopy

or other means, without the permission of the author.

© Shuxiang Yang 2008

Abstract

Uncertainty is inherent in many novel and important applications such as market surveillance, information extraction sensor data analysis, etc. In the recent a few decades, uncertain data has attracted considerable research attention. There are various factors that cause the uncertainty, for instance randomness or incompleteness of data, limitations of equipment and delay or loss in data transfer.

A probabilistic threshold range aggregate ($PRTA$) query retrieves summarized information about the uncertain objects in the database satisfying a range query, with respect to a given probability threshold. This thesis is trying to address and handle this important type of query which there is no previous work studying on.

We formulate the problem in both discrete and continuous uncertain data model and develop a novel index structure, asU-tree (aggregate-based sampling-auxiliary U-tree) which not only supports exact query answering but also provides approximate results with accuracy guarantee if efficiency is more concerned. The new asU-tree structure is totally dynamic. Query processing algorithms for both exact answer and approximate answer based on this new index structure are also proposed. An extensive experimental study shows that asU-tree is very efficient and effective over real and synthetic datasets.

To my beloved family.

Acknowledgements

It is a great time of the two years research in the database group. Now I am grateful to express my gratitude for all who have helped me to make this thesis possible.

First of all, I would like to present my great appreciation to my supervisor, Prof. Xuemin Lin, for his gentleness to offered me such a cherishing opportunity to study under his guard. Without his kind advice, encouragement and insightful direction, this moment will never come true.

I wish to sincerely thank my co-supervisor, Dr. Qing Liu. I will never forget the talk in every Friday night during the first year of my research life. She sacrificed her spare time with her family to lead me to find the right way to do research. I also learned a lot from her enthusiasm, broad knowledge and wise.

I also want to express my gratitude to Dr. Ying Zhang who kept eyes on my research progress closely. He not only taught me valuable skills but also gave me useful ideas, meaningful comments.

Many thanks goes to my fellow students as well: Yi Luo, Muhammad Aamir Cheema, Bin Jiang, Wenjie Zhang, Chuan Xiao, Haichuan Shang. They shared their wonderful research experience with me and made research easier for me. Particularly, I would like to thank Wenjie Zhang who worked closely with me for my thesis topic. My thanks goes to Dr. Wei Wang for his refreshing criticism in our group meeting discussions.

I thank my school, Computer Science and Engineering, for supporting us with sound research environment. I thank every friend in and out Australia, who helped me through every difficulty.

Last but not least, I would like to thank my family, my beloved parents, for their eternal support and love. No matter where I am, no matter what kind of decision I make, they always stand behind me, and try their best to help me.

I dedicate this thesis to all of them.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Since last century, uncertain data has gained substantial amount of research work. The sources of uncertainty are mainly due to various factors such as randomness or incompleteness of data, limitations of equipment and delay or loss in data transfer. Moreover, to keep privacy, uncertainty may be injected manually into the data base.

With the importance and emergence in many traditional and novel applications, such as data integration, sensor data analysis, moving objects, economic decision, market surveillance and trends prediction, etc, uncertain data is catching more and more attention recently.

Consider a database system server which monitors and maintains the locations of moving vehicles in the area of Sydney. Each vehicle U sends its current position periodically in Sydney (for example every 5 minutes). In this case, the server is not able to return the precise location of U at any arbitrary time, because there is no record of the vehicle U in the 5 minutes between the last update and the next one. For example in Figure 1.1, there are 5 records of vehicle U in the database, but we don't know where is U between $time_2$ and $time_3$, because there are 2 paths

from the location at $time_2$ to the point at $time_3$. In that case, uncertainty models are necessary to be introduced to capture the probabilistic location of U.

Extensive research has been done to model uncertain data and execute query over it so far. Research directions include modeling uncertainty [Cod79, GUP06, INV91, Lee92, SBHW06], query evaluation [SD07, CKP03, CKP07, DS04], indexing [TCX+05, CXP+04, LS07, SMP+07, AY08], top-k queries [HPZL08a, RDS, SIC, YLSK, RDS], skyline queries [JPY07, L08], clustering and Mining [KP05, NKC+06, CCKN06], etc. Although *probabilistic threshold range aggregate* query ($PTRA$ query) on uncertain data is very important in practice, this problem still remains unexplored.

1.1 The motivation

Sometimes, we are more interested in the aggregate information of the input objects which satisfy the query, such as the total number of objects in the query region in Figure 1.3 with certain probability. For example, it is more meaningful to figure out how many vehicles currently in the city area of Sydney than to locate every vehicle in the city, if we just want to know whether the traffic condition is busy or not.

1.2 The $PTRA$ Query over Uncertain Data

A range aggregate query (RA query) on certain data returns summarized information about objects satisfying a given query range [TP04]. This type of query is important since users may be interested in aggregate information instead of specific data IDs. aR-tree [PKZT01] is one of the most popular index structure to efficiently answer RA query on spatial space. aR-tree is a modification of R-

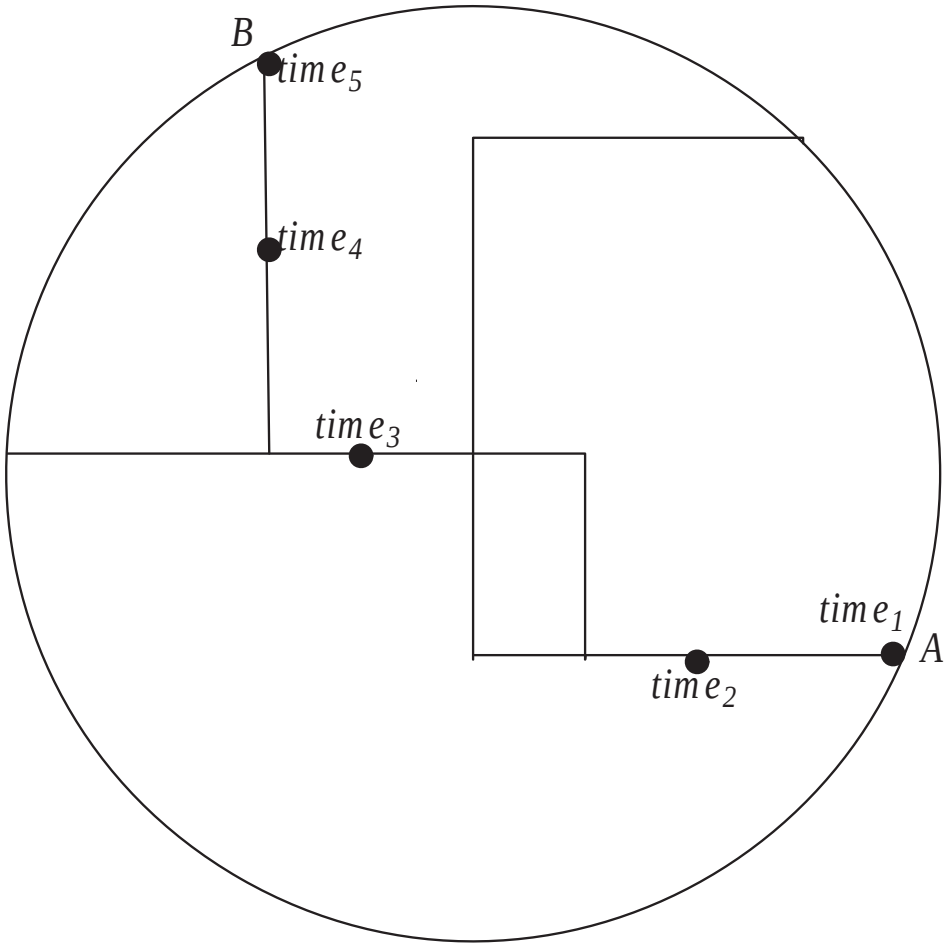


Figure 1.1: The record of locations for vehicle U in Sydney.

tree [Gut84] by storing the number of objects in each entry. Figure 1.2 illustrates the structure of 2-level aR-tree and RA query processing on an aR-tree. Besides R-tree structure information, entries in root node also keep the number of objects contained, such as entry $e_1 \in root$ contains 3 objects. The dashed rectangle is the range of a RA query q. As shown in Figure 1.2, leaf node N_3 is fully covered by q, so N_3 will not be accessed, but adding 3 to the final result. Node N_1 does not need to be accessed either since it does not intersect with q. The only accessed node is N_2 and object O_5 is detected to satisfy q. So the final result is 4.

While many sophisticated techniques have been developed to answer RA query

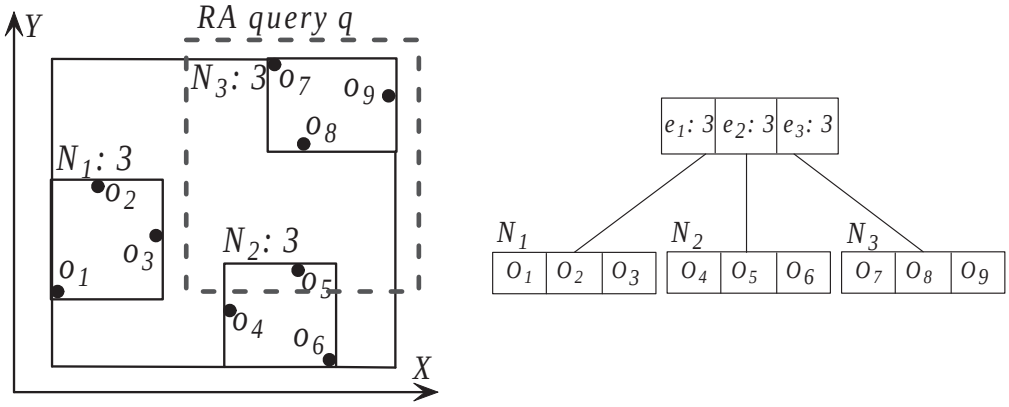


Figure 1.2: Certain Objects Indexed by aR-tree.

over certain data [TP04], the problem of RA query over uncertain data has not attracted much research attention yet. Modeling and answering RA query over uncertain data require comprehensive analysis of probabilities, as shown in Figure 1.3. Assume we still use an aR-tree to index the uncertain objects. A probabilistic threshold range query q fully covers node N_2 and intersects with node N_3. After accessing node N_3, only part of the uncertain object U_7 is in the query range of q. Clearly in this case, we have to retrieve the instances or PDF (Probability Density Function) of U_7 to compute the appearance probability in the region of q, which describes the likelihood that U_7 appears in the region of q. If the appearance probability is no less than the probabilistic threshold, then U_7 will be included in the total number otherwise, it will be excluded from final result. If there are many uncertain objects intersecting with query, this process will be very expensive.

1.3 Exact and Approximate Answer for PTRA Query

As the example shown before, when we try to figure out the current traffic condition in the city area, the precise answer is the number of vehicles. However, it is possible

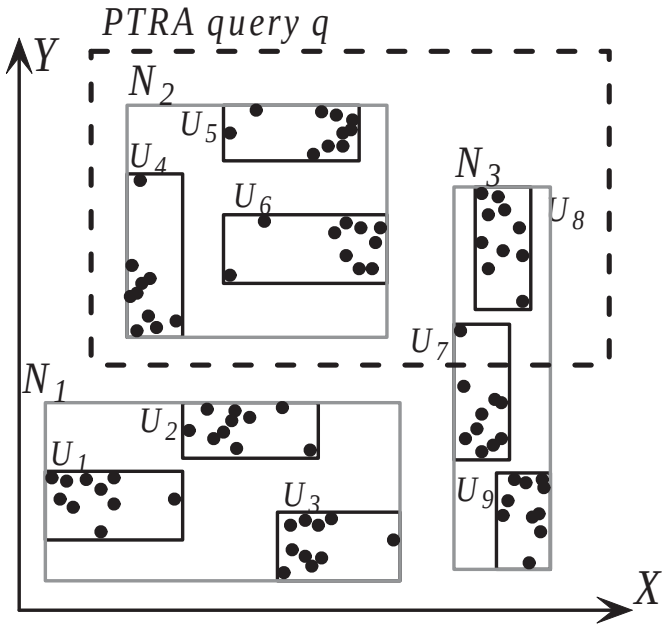


Figure 1.3: Uncertain objects.

that it will consume excessive cost to retrieve the exact number. In that case, it could be quite useful to return an approximate result which keeps the accuracy guarantee as well. It is a very important criterion that a good index structure for uncertain data to process $PTRA$ query should be able to support both the accurate answer and the approximate answer.

1.4 Challenges and Contributions

As shown above, aggregate information retrieval on uncertain objects requires detailed analysis of appearance probabilities from their instances or their $PDFs$. It will be inflexible to compute the appearance probability for every uncertain object to satisfy a probabilistic threshold range aggregate ($PTRA$) query. Effective pruning/validating rules and approximation techniques are necessary to accelerate probabilistic threshold range aggregate ($PTRA$) query processing over uncertain objects efficiently. It is a laborious work to compute the appearance probability

with either instances or $PDFs$, so refining method maintaining the reliability of the performance is often used in this step. The challenges and our contributions are summarized as follows.

1.4.1 Challenge 1: Formalizing The $PTRA$ Query

Although the range aggregate query over certain data has been well studied, this is the first work to address probabilistic threshold range aggregate ($PTRA$) query over uncertain data. It is quite crucial to give explicit definition with appropriate uncertainty model to clearly represent what is $PTRA$ query and what are the specialties of this kind of query.

Our contributions

We formally define range aggregate query over uncertain objects with a given probabilistic threshold which is called $PTRA$ query for short.

As shown before, there are two kinds of models for uncertain data: discrete model, in which each uncertain object contains various number of instances associated with their probabilities, and continuous model, in which $PDFs$ are used to describe the probability distribution in each uncertain object. We provide the problem definition for both models to satisfy different applications. Our problem formulations are quite explicit and we further explain them with simple examples.

1.4.2 Challenge 2: Indexing The Uncertain Data for PTRA Query

As illustrated above, it is very costly to compute the appearance probability of an object. If we use traditional aR-tree based techniques to index uncertain data, the object can be validated when it is covered completely by the query region.

In practical terms, objects are more likely to intersect with query region, in such condition, effective rules to prune/validate these objects without calculating the appearance probability will greatly improve the efficiency.

U-tree [TCX+05] is one of the most popular index structure for uncertain data with series pruning/validating rules aimed to process range query. For our $PTRA$ query, it is not necessary to retrieve every qualified object but a summarized result, so qualifying intermediate entries and supporting approximate answer are important to improve the speed of processing.

Our contributions

A novel index structure, asU-tree is developed to support $PTRA$ query processing. As aR-tree to R-tree, asU-tree is modified based on the U-tree [TCX+05] by storing aggregate information in each entry in the node of asU-tree. Hence if one of the intermediate entries is validated, the aggregate information for the subtree under it can be obtained without accessing the objects contained.

Moreover, to support approximate query processing when efficiency is of more concern, we split the asU-tree into K subtrees and employ dynamic inverse sampling to keep S samples for each subtree to provide approximate result with guaranteed accuracy.

Exact and approximate query processing algorithms are also developed based on asU-tree to answer $PTRA$ query over uncertain objects.

An extensive experimental study over synthetic and real data sets shows that asU-tree supports $PTRA$ query efficiently. Furthermore, sampling based method is not only efficient but also highly accurate.

1.5 Thesis Organization

This thesis is organized as follows. In Chapter 1, we introduce the probabilistic threshold range aggregate query over uncertain data, then we formally give the problem definition in Chapter 2. The related work are introduced in Chapter 3. In Chapter 4, we develop our novel index structure asU-tree in Section 4.1 and the query processing algorithm for exact answer and approximate answer is proposed in Section 4.2. The empirical study is reported in Section 4.3. We conclude the thesis in Chapter 5.

The work presented in this thesis is developed by Prof. Xuemin Lin, Dr. Ying Zhang, Dr. Wenjie Zhang and me. As a main contributor, I formalized the definition of the problem, developed and implemented all the algorithms, and conducted experiments.

Chapter 2

Problem Definition

In this section, we give the formal definition of Probabilistic Threshold Range Aggregate query (PTRA query) over uncertain data for both discrete model and continuous model.

2.1 Discrete Model

In discrete model, each uncertain object U is represented with a set of instances such that each instance $u \in U$ is a point in a d-dimensional numeric space $D = D_1 \times D_2 \times ... \times D_d$ with probability $P(u)$ to appear where $0 < P(u) \leq 1$ and $\sum_{u \in U} P(u) = 1$.

Problem Definition 1. *Given a set of d-dimensional uncertain objects* $\mathcal{U} = \{U_1, ..., U_n\}$*, a query region and a probabilistic threshold* p_q*, a probabilistic threshold range aggregate query (PTRA query) q returns the number of uncertain objects in* $\mathcal{U}$ *satisfying q with probability no less than* p_q*. Formally, the result is*

$$|\{U \in \mathcal{U} | P(U \cap q) \geq p_q\}| \tag{2.1}$$

where $P(U \cap q)$ *is the probability that* U *satisfies* q*.* $P(U \cap q)$ *is defined as the sum*

of probabilities of instances in U in the query region of q. We denote instance u in the query region of q as $u \vdash q$.

$$P(U \cap q) = \sum_{u \in U, u \vdash q} P(u) \tag{2.2}$$

The $P(U \cap q)$ is also called as the appearance probability $P_{app}(U, q)$ of an object U in the query region r_q. ■

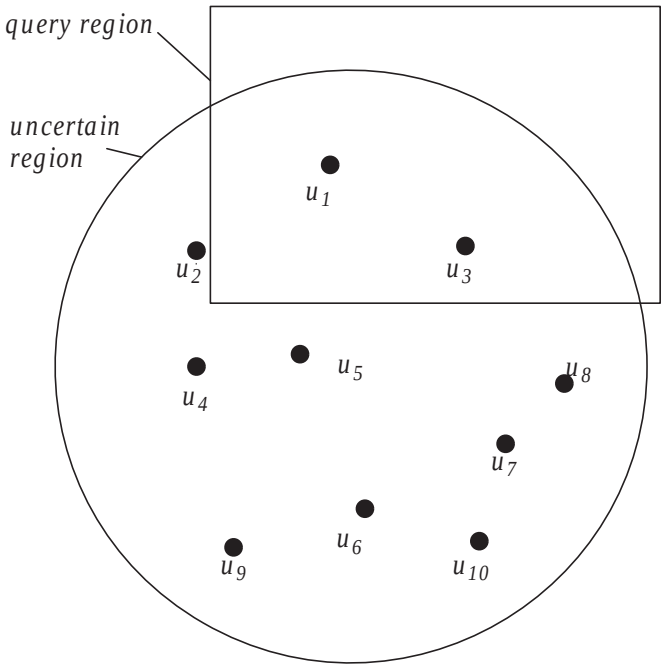


Figure 2.1: uncertain object U in discrete model.

Example 2.1 (PTRA Query in Discrete Model)**.** *As the example illustrated in Figure 2.1, suppose there is only one uncertain object U intersects with the query region, and the object is represented as 10 instances with the same probability to occur, namely* 10%*. Since there are 2 instances from U falls into the region of PTRA query q, so $P(U \cap q) = 20\%$. If $p_q \leq 20\%$, then the result for q is 1; otherwise, the result is 0 (U is excluded from the final result).* ■

2.2 Continuous Model

In continuous model, each uncertain object U is associated with (1) a probability density function $U.pdf(x)$ to describe the probability distribution of U, where x is an arbitrary d-dimensional point, and (2) a d-dimensional uncertain region $U.ur$ to present the appearance region of the center of U.

Problem Definition 2. *Given a d-dimensional query region r_q and a probabilistic threshold p_q, the appearance probability $P_{app}(U, q)$ of an object U in the query region r_q is:*

$$P_{app}(U, q) = \int_{U.ur \cap r_q} U.pdf(x)dx \tag{2.3}$$

Where $U.ur \cap r_q$ represents the intersection region between $U.ur$ and r_q. A probabilistic threshold range aggregate query (PTRA query) q returns the number of uncertain objects in $\mathcal{U}$ satisfying q with probability no less than p_q. The result is in the form:

$$|\{U \in \mathcal{U} | P_{app}(U \cap r_q) \geq p_q\}| \tag{2.4}$$

Where $P(U \cap q)$ is the appearance probability that U intersects with r_q. ■

Example 2.2 (PTRA Query in Continuous Case)**.** *As the example illustrated in Figure 2.2, suppose there is only one uncertain object U intersects with query q, and the PDF of object U follows uniform distribution, then the PDF of object U can be represented as $U.pdf(x) = 1/area(U.ur)$. Following the formula* 2.3*, the appearance probability of U_2 can be calculated as:*

$$P_{app}(U_2, q) = \int_{U_2.ur \cap r_q} U_2.pdf(x)dx = \frac{area(r_q \cap U_2.ur)}{area(U_2.ur)}$$

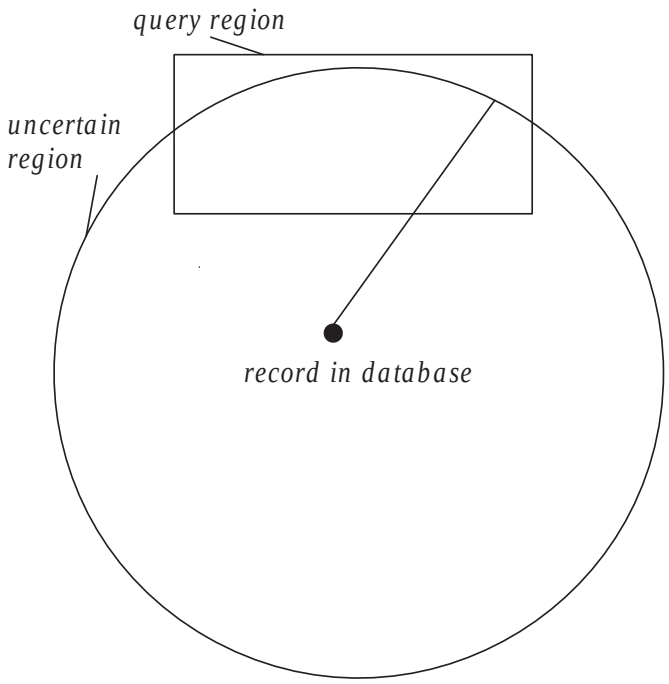


Figure 2.2: uncertain object U in continuous model.

(2.5)

If $P_{U_2} \leq p_q$, then the result for q is 1; otherwise, the result is 0 (U_2 is excluded from the final result). ■

Although we will focus on the continuous case in the next section of this thesis, the idea can be applied to handle the discrete case in similar way. The only difference between discrete and continuous model is the function to compute the appearance probability of uncertain objects.

To make the presentation more clearly for this thesis, a summary of notations is given in table 2.1.

In this thesis, we aim at the solution to handle the $PTRA$ search. The challenges are as follows:

- It is a laborious work to compute the appearance probability for each object.

  It is critical for a system to prune or validate objects, as much as possible,

Notation	Definition
U	uncertain objects
u	instances of uncertain objects
$U.pdf(x)$	the probability density function of U
$P(u)$	the probability of instance u to appear
q	probabilistic threshold range aggregate query
p_q	probabilistic threshold
$P_{app}(U, q)$	the appearance probability of object U in the query region

Table 2.1: The summary of notations.

without calculating the appearance probability.

- In different cases, precision and efficiency have different order of priority based on requirements. A good solution should be able to meet various situations.

Our technique is not only able to get the exact answer for *PTRA* query, but also to give the approximate answer with accuracy guarantee.

To further simplify the process of computing appearance probability of an object, we use the monte-carlo approach in [TCX+05].

Chapter 3

Related Work

In this chapter, we first summarize the literature on uncertain data management to have a comprehensive view on this important area in Section 3.1, then we review the related techniques which inspire our work in the next 5 sections.

3.1 Uncertain Data Management

In this section, we first introduce the uncertainty models in the existing work, and then various query types will be summarized and the related techniques will be introduced as well.

3.1.1 Uncertainty Models

Uncertain data has been studied for dozens of years. A number of models have been proposed to capture the characteristics of the uncertainty. In [ZLPZ08], the authors classified the uncertainty models into three groups:fuzzy model, evidence-oriented model and probabilistic model. Here we focus on probabilistic model which is dominant in the literature in recent years.

In [SBHW06], the probabilistic models have been categorized ranging from incomplete model where every object in the uncertain database has a certain probability independently to complete model. Although complete model [IJ84] is much more powerful because of its ability to represent any probability distribution of the data instances, it is more complicated and infeasible to handle. Normally, we choose an appropriate model for different situations by making tradeoff between the usability and completeness.

In a probabilistic model, an uncertain database D can be in any one of a finite set of states named *possible worlds* [DS04, DS07]. Each possible world appears in certain probability. Formally, an uncertain database consists of a set Ω of possible worlds W. The probability $Pr(W)$ of a possible world $W \in \Omega$ is ranging in $(0, 1]$, and $\sum_{W\in\Omega} Pr(W) = 1$. A query q defines an event. The probability of an event is the sum of the probabilities of all possible worlds in which this event happens. That is, $Pr(p) = \sum_{W\in\Omega, W\models q} Pr(W)$.

We introduce a widely used model called x-relation model [ABS$^+$06, ZLY08] here for its reasonable approximation of the nature of uncertain data.

x-relation model: An uncertain database D is comprised of plenty of independent x-tuples called x-relations. Every x-relation U has one or more alternatives as its instances, and each instance u is associated with probability $P(u)$ representing the probability this alternative being selected, and $\sum_{u\in U} P(u) \leq 1$. Assume that U_1, ..., U_n are the x-relations of D, and W is any subset of the x-relations in D. The probability of W occurring is $Pr[W] = \prod_{i=1}^{n} pw(U_i)$ which:

$$pw(U) = \begin{cases} \text{p(u)} & \text{if } U \cap W = \{u\}; \\ \text{1 - } \sum_{u\in U} p(u) & \text{if } U \cap W = \emptyset; \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

$Pr[W] > 0$, W is denoted as a possible world. Instances from the same x-relation are mutually exclusive, that means at most one of them could appear in a possible world, while instances from different x-relations are independent. There is an example below. Here we use a pair $(u, p(u))$ as instance u with probability $p(u)$:

$x-relations$	
U_1	$(u_1, p(u_1)), (u_2, p(u_2))$
U_2	$(u_3, p(u_3)), (u_4, p(u_4))$

W	$Pr[W]$
$\emptyset$	$(1-p(u_1)-p(u_2))(1-p(u_3)-p(u_4))$
u_1	$p(u_1)(1-p(u_3)-p(u_4))$
u_2	$p(u_2)(1-p(u_3)-p(u_4))$
u_3	$(1-p(u_1)-p(u_2))p(u_3)$
u_4	$(1-p(u_1)-p(u_2))p(u_4)$
u_1u_3	$p(u_1)p(u_3)$
u_1u_4	$p(u_1)p(u_4)$
u_2u_3	$p(u_2)p(u_3)$
u_2u_4	$p(u_2)p(u_4)$

Table 3.1: An example of Uncertain database and possible worlds.

3.1.2 Work on Uncertain Data Analyzing

Zhang *et al*[ZLPZ08] grouped the work on analyzing uncertain data into two big classes based on the types of uncertain data: relational uncertain data and spatial uncertain data.

For relational uncertain data, a lot of research effort has been done on query evaluation [Fuh95, FR97, DS04, DS05, CKP03, CKP07], aggregate queries [CCT96, JKV07, MSS01, MW, RSG05, RB91, SM], join queries [AW, CSP$^+$06] and Top-k queries [HPZL08a, HPZL08b, RDS, SIC].

For spatial uncertain data, the research work has been focused on range

queries [TCX+05, BGK+07, BPS06b], nearest neighbor queries [CCMC08, KKR07], skyline queries [JPY07, L08], similarity joins [KKR07, BPS06a] and clustering and mining [CCKN06, KP05, NKC+06].

3.2 R*-tree and aR-tree

R*-tree proposed in [BKSS90] is one of the most popular index for points and rectangles in spatial database to handle range query, while aR-tree modify R*-tree by adding aggregate information in each entry of R*-tree for aggregate range query. We will introduce the properties of the R*-tree and aR-tree below.

3.2.1 R*-tree

R*-tree is a data structure developed from R-tree [Gut84] which is similar to B-tree. R*-tree splits the data space into hierarchical Minimum Bounding Rectangles (MBR), and it is possible that there are overlaps between MBRs at the same hierarchical level. Figure 3.1 depicts an example of R*-tree which is built from the spatial data.

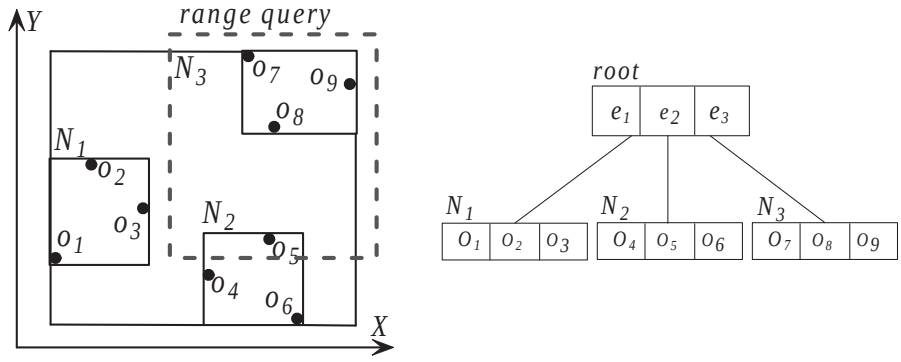


Figure 3.1: An example of R*-tree.

Each node of a R*-tree contains a number of entries (ranged from pre-defined Min number to pre-defined Max number) and each entry in a non-leaf node stores

the address of its child node and the bounding box of all entries within this child node.

R*-tree attempts to minimize :(i) the overlap between two $MBRs$, (ii) the coverage of a MBR, (iii) the margin of a MBR, (iv) the distance between center of two $MBRs$. In the insertion/deletion time, the object will enter/leave the leaf node to make sure the metrics above are always minimized, therefore the closest objects are bundled together in the same leaf node.

In the insertion/deletion time, if the entries overflow the node, the split algorithms will sort the coordinates of the $MBRs$ of the entries to find the split axis and then decide the entry distribution.

At searching time, the query visits the R*-tree from the root and validate the entries contained in the query region. For the node that intersects with the query region but can not be asserted, go further down to the subtree under it, and do the same procedure above recursively. As shown in Figure 3.1, if the entry e_2 in the root can be qualified for it is in the query region r_q, then the object in the child node N_3 will be added to the final answer. The entry e_3 intersects with query region r_q, then its child node will be retrieved and do the same validating processing for the entries in it.

3.2.2 aR-tree

Although the R*-tree is quite efficient for range query, while for range aggregate query in which the aggregate information are needed, a great deal of redundant work has been done. For instance, if the range aggregate query is processed on R*-tree in Figure 3.1, even though the e_2 is qualified, the query window still needs drilling down to the leaf nodes under e_2, and retrieve the objects stored in it to compute the aggregate answer.

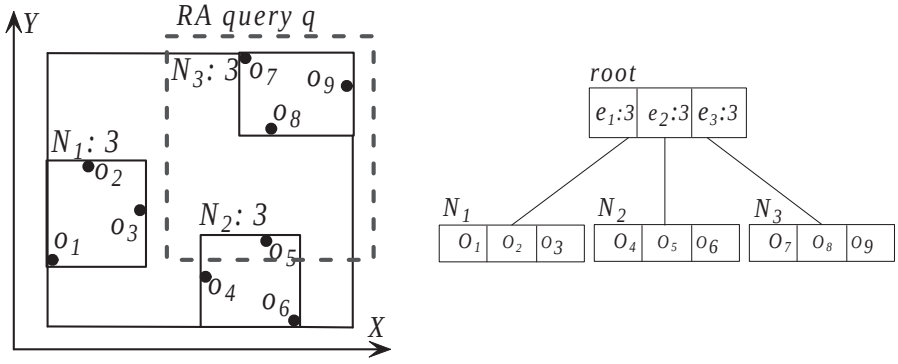


Figure 3.2: An example of aR-tree.

This problem has been handled by aR-tree [JL, LM01, TP04] which adds aggregate information in every entry. There is an example of aR-tree in Figure 3.2 built from the R-tree in Figure 3.1. If e_2 qualified, the aggregate information can be obtained without accessing the nodes under it.

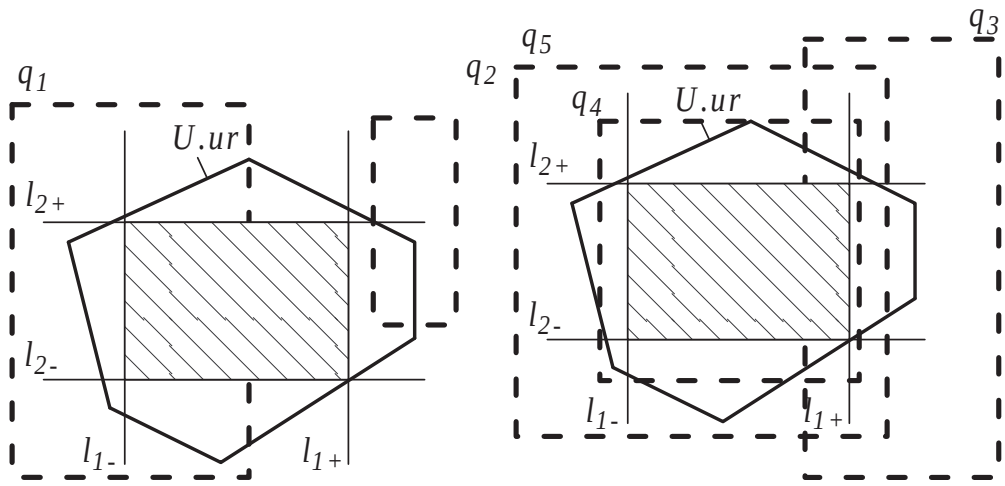
3.3 U-tree

One of the most popular index structure for multi-dimensional uncertain data with arbitrary $PDFs$ is U-tree [TCX+05], which is built based on the structure of R*-tree with a set of pruning and validating rules to support range queries over uncertain data. U-tree is used to prune subtrees that contain no result of a range query, for the leaf entries. Both pruning and validating rules are used to accelerate the query processing.

3.3.1 Probabilistically Constrained Regions

In Figure 3.3, polygon $U.ur$ is the uncertain region of 2-dimensional object U. For a given probability $p_q = 0.2$, in the horizontal dimension, two lines are calculated. U has probability p to occur on the left side of line l_{1-}, also probability p to occur on the right side of line l_{1+}. In the vertical dimension, two such lines are also

computed according to the PDF of U. The portion in these four lines is called probabilistically constrained regions ($PCRs$).



(a) Pruning Case. (b) Validating Case.

Figure 3.3: Pruning/Validating by PCR of Uncertain Object.

Example 3.1 (Pruning Case)**.** *Suppose probabilistic threshold of range query q_1 in Figure 3.3 is 0.8, U can be pruned without accessing the pdf of U since it does not fully contain PCR(0.2). On the other hand, suppose probabilistic threshold of range query q_2 is 0.2, U can not be qualified for the result of q_2 because q_2 does not intersect with PCR(0.2).* ■

Example 3.2 (Validating Case)**.** *Another example explains the validating rules in Figure 3.3. Assume probabilistic threshold of range query q_3 is 0.2, then the object U can be asserted to the result for the probability on the left of line l_{1-} is 0.2 and r_{q3} fully covers the part on the left of line l_{1-}. Similarly, suppose q_4 and q_5 with probabilistic threshold 0.6 and 0.8, respectively, the object is qualified for the both queries q_4 and q_5, because the PCR(0.2) is totally in the query region r_{q4} and r_{q5}.*

■

3.3.2 Conservative Functional Boxes

For different probability values, lines described above are obtained along each dimension. Ideally, U-tree [TCX+05] compute such lines for every probability value to maximize the pruning/validating power. However, this is impossible due to the space limitation. To trade-off between pruning/validating power and space cost, only a set of m probability values are chosen as representatives and their corresponding lines are computed, which are called U-catalog. There are $2 * d * m$ lines kept overall. Conservative functional boxes (CFB) are used to further decrease the size of U-tree. Along each dimension, instead of $2 * m$ values (each line can be determined by one value), a constant number of values 8 are kept for CFB (4 lines, each determined by 2 values). Two lines aim to bound the m PCRs from the "outside" and are called $U.cfb_{out}$. Two lines aim to be contained in the m PCRs, so they are called $U.cfb_{in}$. These two lines further trades-off between pruning power and space consumption. A U-tree is built by organizing the cfb_{out} and cfb_{in} of uncertain objects for all probability values in the U-catalog.

For every intermediate entry e, two d-dimensional rectangles called $e.MBR_{\top}$ and $e.MBR_{\perp}$ are stored in it. Here $e.MBR_{\perp}$ is the MBR of $U.cfb_{out}(p_{min})$ of all objects in e, where p_{min} is the smallest value of the U-catalog. The definition of $e.MBR_{\top}$ is similar but with regard to $U.cfb_{out}(p_{max})$, where the p_{max} is the biggest value of the U-catalog.

Based on $e.MBR_{\top}$ and $e.MBR_{\perp}$, a linear function of p for e is defined as follow:

$$\text{e.MBR(p)} = \alpha - \beta.p \tag{3.2}$$

Where α and β are solved as: $\alpha = e.MBR_{\perp}$ and $\beta = (e.MBR_{\top} - e.MBR_{\perp})/p_m$

.

3.3.3 Dynamic Update Algorithms

Although U-tree has the common tree structure with R*-tree, the optimization metric for update algorithm is different with the one of R*-tree because the entries of U-tree have more complex properties. The optimization metrics of R*-tree in 3.2.1 have been changed with their summed counterparts for U-tree.

For an intermediate entry E, the summed margin is $\sum_{j=1}^{m} Margin(e.MBR(p_j))$ where p_j is the jth value in U-catalog. Similarly the summed overlap and summed center distance are $\sum_{j=1}^{m} Overlap(e_1.MBR(p_j), e_2.MBR(p_j))$ and $\sum_{j=1}^{m} Dist(e_1.MBR(p_j), e_2.MBR(p_j))$, respectively. U-tree aims to minimize the summed metrics.

The insertion/deletion processing for a U-tree is exactly same with R*-tree with the new optimization metrics except the algorithm to split a node which is overflown. For U-tree, they first compute the $e.MBR(p_{\lceil m/2 \rceil})$ of each entry E in the node which needs split, then use the R*-split to decide the entry distribution after splitting.

3.3.4 Query Algorithm

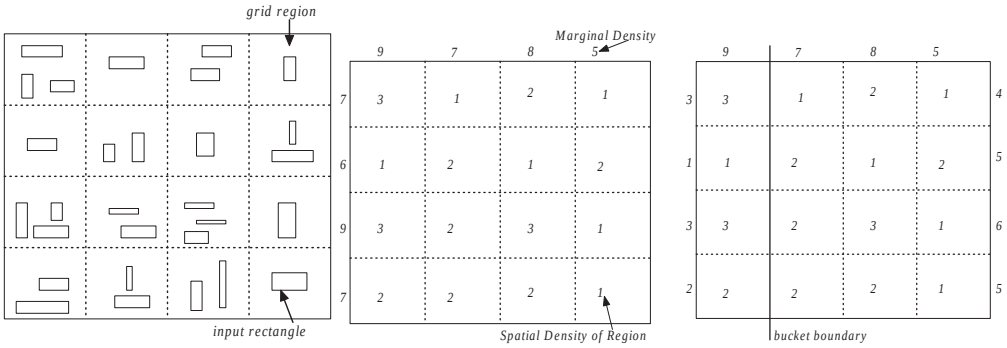
The probability range query algorithm given in [TCX+05] is that the subtree of an intermediate entry e is filtered if query region r_q dose not intersect with $e.MBR(p_j)$, where p_j is the largest value of the U-catalog satisfying $p_j \leq p_q$ and p_q is the probabilistic threshold. When the leaf node is reached, they try to prune or validate every object in it with the pruning/validating rules. For the objects that can not be pruned or qualified, they put them in a candidate set S_{can}, and use the Monte-Carlo(MC) method in the refinement step to compute appearance probability.

3.4 Min-Skew Partitioning

Min-Skew partitioning skill was proposed by [APR99]. This technique is aimed at separating the data space into a number of buckets according to the distribution of input data to approximate spatial data. Min-Skew partitioning has non-ignorable advantages compared with the *Equi* Partitioning skills and tree structure index techniques: first, the splitting is much more reasonable if the space has skew; second, the number of buckets can be controlled at the right one we need.

Two critical notions are proposed in [APR99] to capture the underlying feature of the input data distribution: spatial density of a point representing the number of rectangles that include the point, the other one is spatial skew of a bucket which is the statistical variance of the spatial densities of all points grouped in that bucket to describe the space skew of that bucket.

By Min-Skew partitioning, the input data are grouped into a certain number of buckets, and each bucket is associated with its spatial skew. The objective of Min-Skew partitioning is to minimize the sum of spatial skew of the bucket of the entire input data space.



(a) step I. (b) step II. (c) step III.

Figure 3.4: Min-Skew Partitioning.

Figure 3.4 illustrates the Min-Skew partitioning process: use a uniform grid of regions with the spatial density in each grid to represent the spatial density of the

input data. The process starts from a single bucket including the whole space, if there are less buckets, then split the bucket along the boundary of the grid to reduce the spatial skew mostly. The iteration stops when the buckets reaches the number required.

The empirical study shows that, for small queries, the precision for Min-Skew partitioning is improved by more cells in the uniform grid that are used to approximate data space, however for large queries, the error gets worse with too many regions. To address this problem, they proposed progressive refinement to handle both small and large queries: the algorithm starts from a small number of regions, and for equal intervals of buckets, the regions are refined by splitting each one of them into four. The new regions replace the original ones and properties of the buckets in the algorithm are recomputed using the new regions. The remaining steps are the same as before. In fact, the progressive refinement selectively specifies the buckets which have high-skew regions.

3.5 Dynamic Inverse Sampling

There are many sampling methods [CMN99, CMN98, GM98, Olk, Vit] in the existing literature. A dynamic inverse sampling technique is proposed in [CMR05] aiming at drawing uniform samples from the inverse distribution for data stream [ACc+03, CCD+03, CJSS03] to answer the *I*nverse distribution queries such as Inverse heavy hitters, point queries, range queries [CM05, GM98], inverse quantiles.

3.5.1 Forward Distribution and Inverse Distribution

For data streams, there are many existing solutions to handle forward distribution problems such as samples [CM05] and sketches [MM02], however the inverse distribution problems had not been studied well before [CMR05]. Let's take an example to catch the differences between forward distribution and inverse distribution.

Example 3.3. *Consider the IP traffic on a link as packet p representing* (i_p, s_p) *pairs where* i_p *is a source IP address and* s_p *is the size of a packet. There are two kinds of problem:*

Forward Distribution Problem. *Which IP address sent the most bytes? That is, find i such that* $\sum_{p|i_p=i} s_p$ *is maximum.*

Inverse Distribution Problem. *What is the most common volume of traffic sent by an IP address? That is, find traffic volume W such that* $|i|W = \sum_{p|i_p=i} s_p|$ *is maximum.*

In data stream, if f is a discrete forward distribution over a large set of items X, then inverse distribution, $f^1(i)$, gives fraction of items from X with count i. Inverse distribution is $f^{-1}[0...N]$,

$$\mathrm{f}^{-1}(i) = \text{fraction of IP addresses which sent i bytes} = \frac{|x:f(x)=i, i^{-1}\neq 0|}{|x:f(x)^{-1}\neq 0|} \tag{3.3}$$

$$\mathrm{F}^{-1}(i) = \text{cumulative distribution of } \mathrm{f}^{-1} = \sum_{i>j} f^{-1}(j)[\text{sum of } f^{-1}(j) \text{ above i}] \tag{3.4}$$

3.5.2 The Dynamic Inverse Sampling

In dynamic inverse sampling technique, three arrays are used. *I*tem stores the item from the input; *C*ount stores item counts, namely, how many times this item has

x	1 2 3 4 5
$l(x)$	1 3 2 1 1

Table 3.2: Hash Mapping.

appeared; Unique stores a set of boolean flags to indicate whether the element kept is still unique. Each array is of size $L = \log_{1/r} M$, where r is the ratio to partition input items and M is the range of hash function used. L is also the maximum level number for this data structure. For every item from the data stream, firstly use the hash function to determine its level l in the data structure, then insert it into the corresponding level. If the item ID is the same as *item* in this level, then increase *count* and keep *unique* as *true*; otherwise, increase count and set *unique* to false. When output, from L to 0, output the first *item* and *count* pair with *unique* equals to *true*. Overall, k such data structures are constructed for the pursue of higher precision.

Example 3.4. *Consider the following sequence of elements:*

Input: *3, 1, 5, 4, 2, 1, 1*

Suppose the hash function map the elements to the levels as below:

Figure 3.5 shows the state of the data structure at the update process for insertion. We can find that at time 1, the pair (3,1) returned as sampled value; at time 2, (1,1) and (3,1) are returned; at time 3 and 4, only (3, 1) can be found; from time 5 to 7, both (3, 1) and (2, 1) can be returned. ■

Given a sample from the inverse distribution with size $O(\frac{1}{\epsilon^2} log \frac{1}{\delta})$, Inverse Distribution Queries can be answered with additive error less than ϵ with probability at least 1 - δ.

If we divide the whole data space into enough number of buckets and consider the appearance of an element as an instance and this element as an uncertain object, the dynamic inverse sampling technique can be directly used to answer the

Step	Level 1			Level 2			Level 3		
	item	count	uniq	item	count	uniq	item	count	uniqu
1	0	0	T	3	1	T	0	0	T
2	1	1	T	3	1	T	0	0	T
3	1	2	F	3	1	T	0	0	T
4	1	3	F	3	1	T	0	0	T
5	1	3	F	3	1	T	2	1	T
6	1	4	F	3	1	T	2	1	T
7	1	5	F	3	1	T	2	1	T

Figure 3.5: An example of DIS update process.

PTRA query.

3.6 Monte-Carlo

Monte Carlo methods [PTVF] are a class of computational algorithms used to simulate physical and mathematical systems. Monte Carlo methods compute the result by repeated random sampling. The result of computation relies on good random numbers and its slow convergence to a better precision when having more samples.

Monte Carlo methods follow the general pattern below:

First, define a domain of possible inputs; then generate inputs from the domain

defined randomly; and next perform a deterministic computation on each input; Lastly, aggregate the results of the individual computations into the final result.

Example 3.5. *Let's take an example to use Monte Carlo method to approximate the value of* π*.*

First we draw a circle in a square of unit area, then some small objects with equal area are uniformly scattered over the square. Because the proportion of objects in the circle vs objects in the square is nearly $\pi/4$*, which is also the ratio of the circle's area to the square's area, then we can get a rough number to* π *by counting the number of objects within the circle, multiplying by four and dividing by the number of objects within the square.*

Chapter 4

Index Uncertain Data for PTRA Query

In this chapter, we will introduce our novel technique aiming at handling $PTRA$ queries. This technique is inspired by the previous work we introduced in Chapter 3.

The Construction of Index: Our hierarchical model to index uncertain data is based on the structure of U-tree. For the sake of $PTRA$ queries, we modify the U-tree by adding the aggregate information in every entry, and call this new U-tree as aU-tree as the base index structure. Besides that, another framework is developed on the structure of aU-tree by Min-Skew partitioning and Dynamic Inverse Sampling aiming to support approximate answer. We name this novel structure as asU-tree (aggregate-based sampling-auxiliary U-tree).

The Query Processing: The query algorithms over asU-tree for both exact answer and approximate answer are proposed in this chapter. The exact query processing is based on the aU-tree and similar with the one over U-tree. The approximate query processing over the novel index structure reflects on the crucial gains of our asU-tree.

The index structure of uncertain data for $PTRA$ queries, asU-tree, will be presented with detail in Section 4.1, then in Section 4.2 the corresponding query algorithms will be proposed based on the structure of asU-tree. At the end, in Section 4.3, an extensive empirical study will show the advantages of our technique in various aspects.

4.1 asU-tree

U-tree is a very good index structure for probability range queries over uncertain data. With its pruning/validating rules, U-tree is used to speed up the range query processing on uncertain object, because amount of objects can be filtered without referring to the $PDFs$ or instances of each object U to calculate the exact appearance probability $P_{app}(U, q)$.

However, for the $PTRA$ query, the summarized information is required, therefore it seems no necessary to access the objects in the leaf nodes if they can be asserted in the upper node. For example, in Figure 4.1, node N_2 is totally in the $PTRA$ query region, so the objects contained in N_2 are obviously qualified for the query, we only need to add the number of objects stored in this node to the answer and stop here digging down the subtree under it. In the case above, if use U-tree directly to process our $PTRA$ queries, plenty of redundant work has to be done because aggregate information is not available in the intermediate entries and moreover U-tree is not able to validate intermediate entries as well.

On the other hand, sometimes we are more interested in the approximate answer. For example, we are planning to cross the downtown, and wondering if the traffic is busy or not in that area right now. At that time, it is luxurious to search the exact number of vehicles in downtown and an appropriate approximate num-

ber is enough for our request. Another deficiency of U-tree for our $PTRA$ queries searching is that U-tree does not support approximate query processing.

From the observations above, based on the structure of U-tree, and inspired by Min-Skew partitioning and Dynamic Inverse Sampling, we construct a novel structure, asU-tree, to handle the $PTRA$ queries with high accuracy and efficiency guarantee. Compared with U-tree, the following improvements are made for asU-tree:

- Aggregate information is added in every entry of U-tree. This information can be updated whenever we insert or delete an object in the U-tree.
- Creative rules are developed for validating intermediate entries to accelerate the $PTRA$ query processing.
- Another framework is constructed on the structure of aU-tree: splitting the whole tree into K subtrees, and doing Dynamic Inverse Sampling for each subtree to support the approximate query processing.
- Exact query processing and approximate query processing are proposed with efficiency and accuracy guarantee, respectively.

4.1.1 Index Overview

The original U-tree is used to prune the intermediate entry that contains no result. For the intermediate entry that can not be pruned, we have to drill down the subtree under it, in the worst case, to the leaf nodes to verify the uncertain objects contained in it. For the objects that can not be eliminated or asserted by the rules, we have to compute the appearance probability with the $PDFs$/instances. In fact, as the example showed before, the intermediate entry E can be validated

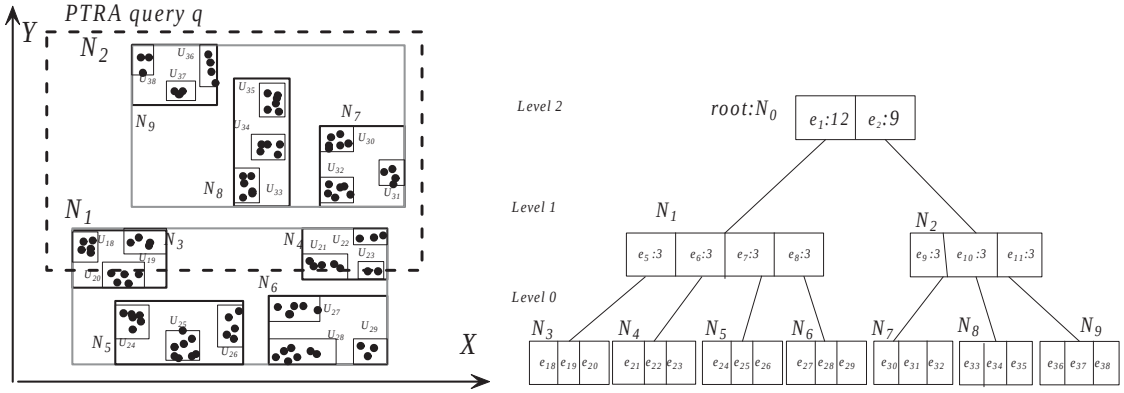
if it is totally covered by the query region r_q, thus we only need to step down the intermediate entry which intersects with the query region r_q but can not be pruned or qualified.

From the observation above, for our asU-tree, we add aggregate information in every entry of U-tree. Similar with the modification of aR-tree to R-tree, the aggregate information can be updated dynamically. With this improvement, when the intermediate entry can be qualified, the aggregate information is adopted directly in this entry without accessing the subtree under it. This modification can save the amount of cost, especially when the query has broad searching region.

The asU-tree shares same tree structure with U-tree: The tree is built in a bottom-up manner. The leaf nodes are on level 0, and the root is on level L, where L is the height of the tree. Each node N occupies a disk page size which records: (i)the disk address of N; (ii)the entries in N. Each entry E in a leaf node corresponds an uncertain object U and it keeps: (i): $U.pcr(r)$; (ii): $U.pdf$. For the entry E in non-leaf node, we keep: (i): a pointer to its child node; (ii): $e.mbr(C_1)...e.mbr(C_m)$, where $e.mbr(c)$ is the MBR of $e_1.mbr(c)...e_f.mbr(c)$ and $e_i...e_f$ are in the child node of E; (iii): $e.sl(C_1)$ ... $e.sl(C_m)$, where $e.sl(c)$ is the smallest value in $e_1.sl(c), ..., e_f.sl(c)$ and sl means side length.

To support approximate answer, we divide the asU-tree into K non-overlapping subtrees by picking up K nodes of the asU-tree as the representatives of the asU-tree, and we call these K nodes as Best K Nodes(BKN). These K subtrees cover the whole data set without overlap and each of them keeps S objects randomly sampled from the subtree it represents. We use a virtual node for each BKN to keep the samples selected from the subtree under it, and call the virtual node as its Sampling Node. Now the whole asU-tree is condensed by the K Sampling Nodes. When a query region r_q reaches the BKN and this BKN can not be pruned or

qualified, the query region r_q goes to its Sampling Node rather than digs down the subtree under it.



(a) Uncertain Objects. (b) asU-tree.

Figure 4.1: Uncertain Objects and its asU-tree.

The ideal case is that given certain number of K, each BKN stands for one node of an asU-tree. However, in most cases, the BKN is not always ready-made in the asU-tree. Take the Figure 4.1 as an example, there are 2 nodes on level 1 and 7 nodes on level 0, if we want to divide the asU-tree into 4 subtree, we have to merge the nodes on level 0 or split some node on level 1 for there is no combination of the nodes to get $BKNs$. At that time, Min-Skew partitioning skill is applied to solve this problem.

The Figure 4.2 illustrates the structure of an asU-tree, where K is 3.

In the following subsections, we will introduce how to construct asU-tree and process $PTRA$ queries on it. After that, the advantages of asU-tree will be shown.

4.1.2 Aggregate Information Integration

As said before, to answer the $PTRA$ query, it will be much more efficient if the aggregate information can be obtained at hand whenever the intermediate entry has been qualified, because it saves considerable number of unnecessary work to

access the subtree under that intermediate entry to retrieve the objects stored in it for the aggregate function to calculate the result.

Similar with the adjustment of aR-tree to R-tree for the range aggregate query over certain data, U-tree is modified by embedding aggregate information in every entry of U-tree for requirement of $PTRA$ query. We call this modified U-tree as aU-tree.

With aU-tree, if the intermediate entry is totally covered by the query range, the aggregate result for the subtree underneath can be released immediately without accessing every object in it.

The process to update the aggregate information for aU-tree is similar as the one for aR-tree: whenever insert or delete an object, besides rectifying the correlative information of U-tree, the aggregate information of the corresponding entries are updated automatically as well.

4.1.3 Best K Nodes Construction

Compared with U-tree, aU-tree makes a great improvement to save considerable labor for the $PTRA$ queries. However, the structure of aU-tree is still not able to support the approximate answer. The most creative part of our technique is that the aU-tree is further modified to support approximate query processing. The intuitive idea is that we divide the aU-tree into K non-overlapping subtrees, and for each subtree we keep S objects which are sampled from the objects stored in this subtree. If the subtree can not be pruned or qualified, the query range visits the samples rather than drill down into the subtree. The procedure to check if the samples qualify the query or not is the same as the one to check the normal uncertain objects in U-tree: prune or validate it using the proposed rules, if it can not be eliminated or qualified, put it to the S_{can} set to calculate the appearance

probability by the refinement method.

The Obstacles to Locate BKNs

The ideal situation is that we separate the aU-tree into K non-overlapping subtrees by picking up K nodes called $BKNs$ on the aU-tree. For example in Figure 4.1, we can take N_1, N_7, N_8, N_9 as the $BKNs$ to split the whole aU-tree into 4 subtrees. However, the situation becomes obscure if we attempt to divide the aU-tree into some 'inconvenient' number of subtrees. Take another example in Figure 4.1: if we attempt to make 6 subtrees, which group of nodes should be chosen as the $BKNs$ from N_1, N_2, N_3, N_4, N_5, N_6, N_7, N_8, N_9. How about if we attempt to make 8 subtrees?

If we consider the objects in U-tree as normal spatial rectangles, the U-tree can be treated as R*-tree. As pointed out in [APR99], there are some defects of R-tree family structure to index the spatial rectangles, such as the insertion algorithm didn't take the spatial skew as one of the optimization metric to construct the R-tree, so the spatial skew of the nodes that split the data space is unknown. Another defect is that it is impossible to control the exact number of entries that further separate the node they are stored in, when the R-tree is being built by inserting the objects.

The U-tree has the same problems with the R-tree to index the spatial data. It is impossible to choose the $BKNs$ before finishing the construction of U-tree because we are unable to control the number of nodes on each level of U-tree. Set the number of input objects as Num, the maximum number of entries in a node of U-tree as Max, and the minimum number as Min, then on level L, the total number of entries Ent is between $\frac{Num}{Max^L}$ and $\frac{Num}{Min^L}$. It is hard for us to fix the exact number of entries in each node, therefore it is not convenient to locate the nodes

as $BKNs$ on the aU-tree to separate the aU-tree into K subtrees when the U-tree is still under construction.

From the analysis above, we can see that the main difficulty lies on how to locate the $BKNs$ on an aU-tree structure and how to verify the $BKNs$ that we choose is the best option.

BKNs Construction

A novel tuning skill is proposed here to split aU-tree into arbitrary number of subtrees automatically, and it does not depend on the number of entries in every node. This skill is inspired by the fundamental work of Min-Skew partitioning proposed in [APR99].

To construct the arbitrary K subtrees from the aU-tree, the splitting criteria to ensure the nodes we choose are the best ones to separate the aU-tree: the sum of the spatial skew of the K subtrees should be minimized. The notion of spatial skew is defined by [APR99] as we introduced in Chapter 3.

If we take the $MBRs$ of the objects as the initial grid of regions in Min-Skew partitioning algorithms, the most straightforward method following the original algorithm to divide the aU-tree into K subtrees is: we group those $MBRs$ of the uncertain objects into K buckets which lead the greatest reduction of the total spatial skew. Take Figure 4.1 as an example, in order to form 3 subtrees, we compute the spatial skews for N_3, N_4, N_5, N_6, N_8 and N_9, then find that it will reduce the spatial skew mostly if N_3 and N_4 are grouped together, N_5, N_6 are put into same bucket, and the others are in the same bucket. Therefore we get the 3 $BKNs$ as shown in Figure 4.2.

The special situation which is different from the traditional Min-Skew partitioning is that we do Min-Skew partitioning on an aU-tree which already is a hierarchial

index structure of uncertain objects rather than do it on the spatial space. In an aU-tree, the nodes on level L can be considered as the boundary dividing the aU-tree into S non-overlapping subtrees($\frac{N}{Max^L} <= S <= \frac{N}{Min^L}$). Following the optimization metric for aU-tree, on each level of aU-tree, the closest objects have been bounded together in the same node.

Based on the observation above, we give a progressive approach to reduce time and space consumption to construct K subtrees using Min-Skew partitioning. Instead of doing Min-Skew partitioning from the root of the aU-tree, we first locate the right level L where the number of nodes is equal to or less than K, but the number in the lower level is greater than K. If the number of nodes on level L is equal to K, these K nodes are the $BKNs$ we are looking for; otherwise, split each node N_i on level L into G_i buckets and the sum of the buckets G_i equals K, that is $\sum_{G_i \in N_i} G_i = K$. The number of buckets G_i of node N_i depends on the portion of spatial skew of N_i . Set the skew of N_i as S_i, then $G_i : S_i = G_j : S_j$. We put the nodes in the same bucket together as one of the $BKNs$.

Example 4.1. *As illustrated in Figure 4.2, suppose we attempt to split the aU-tree into* 3 *subtrees, first we locate at level* 1 *which have* 2 *nodes* N_1, N_2 *and* 7 *nodes on level* 0*, then compute their skew* S_1, S_2, *respectively, if* $S_1 : S_2 = 2 : 1$, *then we split* N_1 *into 2 buckets* B_{11} *which contains* N_3 *and* N_4, B_{12} *which contains* N_5 *and* N_6 *by Min-Skew partitioning skill. Now we get the* 3 *subtrees represented by the* 3 *buckets* B_{11}, B_{12}, B_2, *and the nodes in the same bucket are considered as the same* BKN. *For example* N_3 *and* N_4 *are taken together as one of the* $BKNs$. ■

When the entry needs further split, the Min-Skew partitioning will be applied on its child node.

The algorithm to apply Min-Skew partitioning is the same with the one in [APR99] except that we use the $MBRs$ of entries in the node which will be

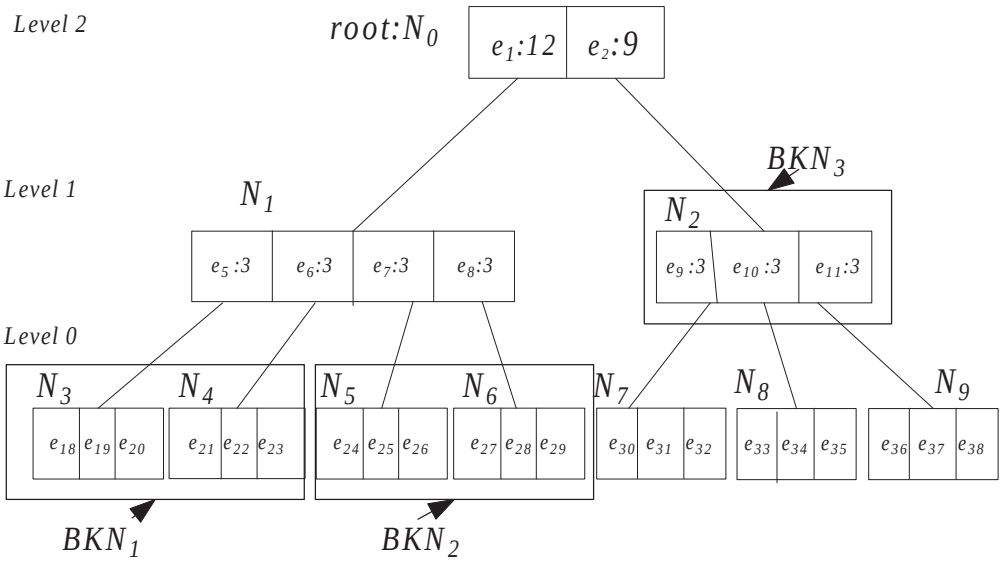


Figure 4.2: Best K Nodes Selection

Algorithm 1: SplitEntry(E, B)

Input : The entry needs to be split E; The number of bucket B

Output: The buckets $Buckets$

1 get the child node N of E;

2 MinSkew(N,B);

split as the uniform grid of regions in the original algorithm. The pseudo-code for this step is provided in algorithm 2:

If the $BKNs$ are not available from the existing nodes of asU-tree, we have to split some node into different number of buckets. The algorithm 3 illustrates the process to find which node needs to be split and how many buckets it should have.

4.1.4 Sampling Auxiliary

After the aU-tree has been divided into K subtrees, we start to apply an appropriate sampling technique to condense each subtree to be a smaller tree. As we introduced before, if we treat an object as an item and an instance of the object as

Algorithm 2: MinSkew(N, B)

Input : The node N; The number of buckets B required

Output: The buckets $Buckets$

```
1 while the number of bucket BT ≤ B do
2     for each current bucket do
3         compute the spatial skew of the bucket;
4         split point along its dimensions that will lead the maximum reduction in
          spatial skew;
5     for the bucket whose split reduces the spatial skew greatest, split the bucket
      into two;
6     assign regions from the old buckets into the new ones;
7 return the buckets Buckets
```

the appearance frequency of the item, Dynamic Inverse Sampling (DIS) can be applied directly to select S samples for each subtree. However, to make it meaningful to keep the appearance probability of the object, further splitting the subtree into smaller subregions, such as uniform grids, has to be done. However, it will take much more storing space to keep the information of subregions for every subtree. Enlarging the storing space for each subtree means enlarging the pagesize for U-tree. The pagesize is linear with time to construct aU-tree and process query, so we would better avoid making further space division for the subtree to bring more information which needs to be kept.

According to the original DIS algorithm, three arrays should be kept: *item* storing the ID of item from the input, *count* storing the appearance of the item, *uniq* is the boolean flag indicating if this item is still unique. Here, we just need to keep two arrays because the appearance probability of any object in the subtree

Algorithm 3: Construct $BKNs$

Input : The aU-tree U

Output: The $BKNs$

```
1  L = ⌊log(K)/log(C)⌋;
2  Entry.Level = L count the spatial skew of this entry SK;
3  insert the entry into list List according to SK in decreasing order;
4  if the number of entries in the List equals K then
5      assign List to BKNs;
6      return BKNs
7  else
8      for each E ∈ List do
9          Compute the number of buckets B for E;
10         SplitEntry(E, B);
11         put the Entries in each buckets into BKNs;
12     return BKNs
```

equals one at all time, then the *count* array can be eliminated.

For approximate answer, the precision is in direct proportion to the number of samples, while the efficiency is in inverse proportion to the number of samples. We will test the ideal number of samples in the empirical study. The samples will be inserted in a new node for the subtree called Sampling Node.

The remaining step is to apply DIS in the same manner as explained in [CMR05]: keep desired size of data structure as we introduced before for each subtree, and use hash function to map the objects stored in the subtree to different levels with *uniq* to imply that the object is still the unique one in this level or not. At the output time, we choose the unique object for each data structure as the sample and insert it to the Sampling Node. This procedure will halt when the required size of samples is obtained.

After DIS, small number of objects are sampled uniformly for each of the K subtrees that are represented by the $BKNs$. Now the asU-tree, which stands for aggregate based sampling auxiliary U-tree, has been constructed completely.

4.1.5 Dynamic Update of asU-tree

Whenever an uncertain object is inserted into or deleted from the asU-tree, three sets of information need to be updated. Take insertion operation as an example and deletion operation is similar.

Dynamic Insertion/Deletion Procedure: Firstly, the object is inserted into asU-tree as described in [TCX+05] except that the aggregate information for each entry in each node of the asU-tree needs to be updated as well.

Dynamic Splitting Procedure: If there is any change in the node distribution, the Min-Skew partitioning will be applied as described in 4.1.3 to adjust the location of $BKNs$ based on the renewed node distribution of the asU-tree.

Dynamic Sampling Procedure: Lastly, if the $BKNs$ are changed, the dynamic inverse sampling will be applied for the new $BKNs$, otherwise the corresponding dynamic inverse sampling structure for all $BKNs$ will be updated as described in [CMR05].

Algorithm 4: Dynamic Update(T)

Input : The asU-tree T

Output: The updated asU-tree T

```
1 insert/delete objects;
2 if any node split occurs then
3     Construct BKNs;
4     for each new BKN do
5         do DIS;
6 else
7     for each BKN do
8         update DIS;
9 return the updated asU-tree T
```

4.2 Query Processing Algorithms

With our asU-tree, both exact and approximate queries can be executed with high efficiency and effectiveness. In fact, the exact $PTRA$ query processing and the proximate $PTRA$ query processing on asU-tree have many similarities with the probability range query processing on U-tree, because asU-tree shares the same foundation with U-tree. The major difference between the two kinds of queries over uncertain data lies in their different goals. The $PTRA$ query is aimed to

find the aggregate information for the whole data set instead of retrieving every object which is qualified for the query. To answer $PTRA$ queries, based on the rules in [TCX+05], we proposed the rules to eliminate/validate both subtrees and objects on asU-tree for $PTRA$ queries.

The rules to prune/validate subtrees for $PTRA$ query q with search region r_q and probability p_q are given as following:

- Rule 1: For $p_q > 1 - p_m$, an object U can be pruned if query region r_q does not totally cover $U.cfb_{in}(p_j)$, where $p_j (1 \leq j \leq m)$ is the smallest value in the U-catalog not less than $1 - p_q$;

- Rule 2: For $p_q \leq 1 - p_m$, an object U can be eliminated if query region r_q does not intersect with $U.cfb_{out}(p_j)$, where p_j is the largest value in the U-catalog not greater than p_q;

- Rule 3: An object U can satisfy query q if query region r_q fully contains the part of $U.MBR$ between planes $U.cfb^{i-}_{out}(p_j)$ and $U.cfb^{i+}_{out}(p_j)$ for some $i \in [1, d]$, where p_j is the largest value in the U-catalog not greater than $(1 - p_q)/2$;

- Rule 4: For $p_q > 0.5$, an object U can be validated if the query region r_q completely covers the part of $U.MBR$ on the right (or left) of $U.cfb^{i-}_{out}(p_j)$ (or $U.cfb^{i+}_{out}(p_j)$) for some $i \in [1, d]$, where p_j is the largest value in the U-catalog not greater than $1 - p_q$;

- Rule 5: For $p_q \leq 0.5$, an object U can be qualified when query region r_q totally contains the part of $U.MBR$ on the left (or right) of $U.cfb^{i-}_{in}(p_j)$ (or $U.cfb^{i+}_{in}(p_j)$) for some $i \in [1, d]$, where p_j is the smallest value in the U-catalog not less than p_q;

- Rule 6: For p_q, the subtree of an intermediate entry e can be pruned if query region r_q does not intersect $e.MBR(p_j)$ (for some $j \in [1, m]$), where p_j is the largest value in the U-catalog satisfying p_j not greater than p_q;

- Rule 7: For p_q, the subtree under an intermediate entry e can be validated if the $e.MBR$ inside r_q.

For the object in the candidate set S_{can} which is required to refer to the PDF/instances of it to calculate the appearance probability to determine if it belongs to the result or not, we follow the refinement step in [TCX+05] using Monte-Carlo method. The asU-tree does not make any innovative improvement to calculate the appearance probability, since its contributions in the novel index structure for uncertain objects.

4.2.1 The Refinement Step

When the uncertain object U can not be pruned or validated, the appearance probability p_{app} of it in the query region r_q needs to be calculated and compared with the probabilistic threshold p_q. It is not hard to get the appearance probability p_{app} if the uncertain region $U.ur$ and query range r_q are both in regular shape and the probability distribution is described in integrated function $PDFs$. For example, for any uncertain object U from the input if $U.ur$ and r_q are rectangles and $U.pdf$ describes a uniform distribution, the appearance probability $p_{app}(U, p)$ can be simply computed by the ratio between the part of $U.ur \cap r_q$ and r_q.

However, the situation is much more complicated in reality because normally the probability distribution is irregular and the intersection area between $U.ur$ and r_q is not always rectangles but some complex shape. At that time, it is very costly to compute the exact appearance probability p_{app}. The Monte-Carlo(MC) method

is introduced here to remedy the problem.

In MC method, a function $f(x)$ is used to represent $U.pdf(x)$ for a d-dimensional point x in $U.ur \cap r_q$. The query region r_q is represented by r_{mbr} which is the MBR of the r_q, then a number of uniform points are randomly generated in the area $r_{mbr} \cap r_q$. Set these points as $x_1, ..., x_m$, respectively, the appearance probability can be estimated with accurate guarantee if the number n of points is big enough in the following equation:

$$P_{app}(U, q) = vol.\frac{1}{n}\sum_{i=1}^{n} f(x_i) \tag{4.1}$$

Where vol denotes the volume of $r_{mbr} \cap r_q$.

4.2.2 Exact Query Processing

When accurate result for the $PTRA$ query required, the query q traverses the asU-tree from the root with Rule 6 and Rule 7 to prune/qualify the entries in the root. If the intermediate entry can be asserted, the aggregate information in this entry is contributed to the final result and the subtree under it is not going to be further visited. For the entry which can not be eliminated or validated, we retrieve its child node and do the same procedure for the child node recursively. When the leaf node is reached, we will try to prune/validate its entries which are the uncertain objects with Rule 1 to Rule 5. If the object can not be neither pruned nor qualified, we send it to the candidate set S_{can}. After processing all necessary nodes of the asU-tree, we start to apply the tuning step for the objects in the set S_{can} as described in 4.2.1. The pseudo-code for exact query processing is given in algorithm 5 and algorithm 6:

Example 4.2. *Take an example to show the exact query processing in Figure 4.3:*

Algorithm 5: ProcessNodeForExactResult(N, *Stack*, Result)

Input : The asU-tree node N; storing stack *Stack*; final result *Result*

Output: storing stack *Stack*; final result *Result*

```
1 for each entry E_c in N do
2     if the entry E_c is qualified then
3         add the aggregate information of E_c to Result;
4     else if E_c can not be pruned then
5         put E_c to Stack;
```

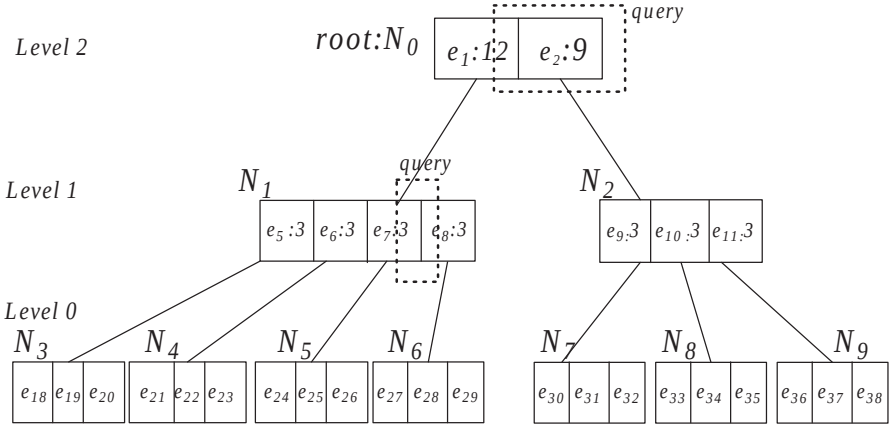


Figure 4.3: Exact Query Processing

The query visits the asU-tree from the root which contains Entries E_1, E_2 *and the query region* r_q *fully covers* E_2, *intersects with* E_1. *The aggregate result* 3 *from* E_2 *is added to the final result that is 3 now and the query region* r_q *stops to go down the subtree under* E_2, *while* E_1 *can not be disqualified by rule* 6 *and needs to be further explored, then we retrieve the child node* N_1 *of* E_1.

In node N_1, *the entries* e_7 *and* e_8 *are intersected with query region* r_q *and still can not be pruned or validated, then we retrieve their child node* N_5 *and* N_6 *which are leaf nodes and contain uncertain objects. Now there are 6 objects need to be checked totally. Suppose we process the objects in* N_5 *and* N_6 *one by one and find*

Algorithm 6: Exact Query Processing(T, r_q, p_q)

Input : The asU-tree T; the query region r_q; the threshold p_q

Output: The aggregate result *Result*

```
1  initialize the stack Stack for storing entries;
2  initialize the final result Result;
3  get the root node R of asU-tree;
4  ProcessNodeForExactResult(R; Stack; Result);
5  while Stack is not empty do
6      take the first entry E in Stack;
7      if E is not in the leaf node then
8          get the child node N of E;
9          ProcessNodeForExactResult(N, Stack, Result);
10         pop E from Stack;
11     else
12         if E can be validated then
13             add the aggregate information to Result;
14         else if E can not be pruned then
15             send it to the candidate set S_can;
16         pop E from Stack;
```

that there are 3 objects qualified, then the final answer is 6. ■

4.2.3 Approximate Query Processing

It is the biggest contribution of asU-tree to support approximate query processing for $PTRA$ queries with accurate and efficient guarantee.

The approximate query processing has no difference with the accurate query processing from the beginning step until starting to process the BKN. When the BKN is reached, instead of doing the recursion to retrieve the subtree of the BKN, the query visits the Sampling Node of the BKN to estimate an approximate answer.

There are S samples which are selected from the objects stored in the subtree of BKN by DIS in the Sampling Node. We prune/validate the samples with Rule 1 to Rule 5, for the sample which can not be pruned or validated, apply the refinement step to check if it belongs to the answer. After accessing every sample in Sampling Node, the approximate result for this subtree is valuated as:

$$R = M * \frac{N}{S}; \tag{4.2}$$

Where R represents the approximate result for this subtree, M is the number of samples that satisfy the query q, and N is the number of total objects in this subtree.

It needs to be pointed out that if the Sampling Node is built from more than one intermediate entries, once one of these entries processed, the others in the same Sampling Node should not be accessed any more to avoid repetition of calculation.

Example 4.3 (Avoiding Repetition)**.** *Let's specify the procedure to deal with the Sampling Node which consists of more than one entries with an example in Fig-*

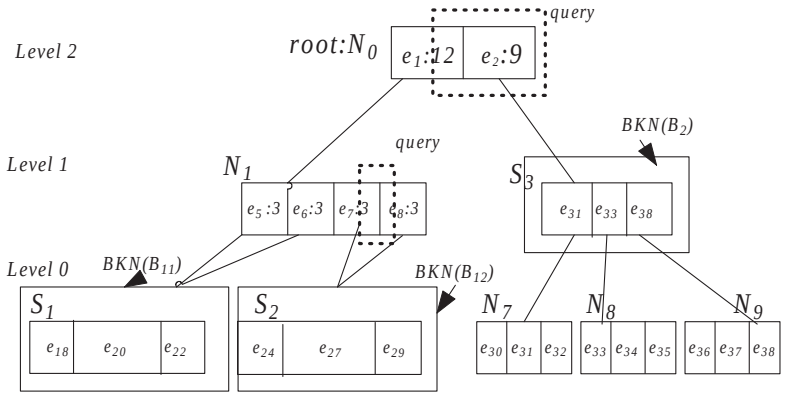


Figure 4.4: Approximate Query Processing

ure 4.4. The entry e_2 in the root is qualified and its aggregate information 3 is added to the result, while entry e_1 can not be neither asserted nor eliminated by the asU-tree rules, and needs to be further explored, then we retrieve the child node N_1 of e_1.

For the node N_1, we access its entries in it and find that entry e_7 is selected in the bucket as one BKN with other entry e_8 in N_3, then we visit the Sampling Node S_2 to check the objects sampled in it. There are 3 samples stored in S_2 which are selected from the 6 objects in entry e_7 and entry e_8, and we just check these 3 objects. If there are 2 samples qualified, the answer for the subtree which is represented by S_2 is: $2 * \frac{6}{3} = 4$, and then we get the final answer which is: $3 + 4 = 7$.

After accessing Sampling Node S_2, the entry e_8 is eliminated from processing queue because it is marked as the other one in the same bucket with e_7. ■

We give the pseudo-code for approximate query processing in algorithm 7 and algorithm 8:

Example 4.4 (Approximate Query Processing)**.** *Let's use an example in Figure 4.4 to illustrate the approximate query processing: The procedure starts from the root node R: prune or validate the entries e_1, e_2 in the root. Obviously, e_2 is validated for it is totally in the query region r_q, then the aggregate information 3 in e_2 is*

Algorithm 7: ProcessNodeForApproximateResult(N; *Stack*; *Result*)

Input : The asU-tree node N; storing stack *Stack*; final result *Result*

Output: storing stack *Stack*; final result *Result*

```
1 for each entry E in N do
2     if E is qualified then
3         add the aggregate information of E to Result;
4     else if E can not be pruned then
5         if E is marked as BKN which is not processed then
6             return approximate result R for E from Sampling Node N_s;
7             add R to the final result Result;
8         else if E is not BKN then
9             put E to the stack Stack;
```

added to the final result. if entry e_1 can neither be pruned nor qualified, and it is not one of the $BKNs$, then we retrieve its child node N_1 and put the entries in N_1 to the stack. we start from the first entry in the stack to apply the rules to prune or validate.

Suppose entry e_7 is marked as one of the $BKNs$, then we go to its sampling node S_2 rather than its child node and check the samples in S_2. Assume there are 2 objects qualified in the 3 samples that are selected from 6 objects, then the result for the subtree under e_7 is: $R = 2 * \frac{6}{3} = 4$, so we add R to the final result which is 7 now. Then we fetch the last entry e_8 in the stack and find that it has been marked as the same $BKNs$ with e_7, so the final approximate result is returned as 7.

■

Algorithm 8: Approximate Query Processing(T, r_q, p_q)

Input : The asU-tree T; the query region r_q; the threshold p_q

Output: The aggregate result *Result*

```
1  initialize the stack Stack for storing entries;
2  initialize the final result Result;
3  get the root node R of asU-tree;
4  ProcessNodeForApproximateResult(R; Stack; Result);
5  while Stack is not empty do
6      take the first entry E in Stack;
7      if E is not in the leaf node then
8          get the child node N of E;
9          ProcessNodeForApproximateResult(N; Stack; Result);
10         pop E from Stack;
11     else
12         if E can be validated then
13             add the aggregate information to Result;
14         else if E can not be pruned then
15             send it to the candidate set S_can;
16         pop E from Stack;
```

4.3 Experimental Analysis

We present a thorough study on the performance of our techniques in this section. All algorithms are implemented in C++. Experiments are run on PCs with Intel P4 2.8GHz CPU and 2G memory under Debian Linux. The disk page size is fixed to 8192 bytes.

The asU-tree is updated from the U-tree to index uncertain objects for $PTRA$ query. There is no previous work aiming at this kind of query, for comparison, we implement parallel experiment on both asU-tree and U-tree using the same data sets in [TCX+05]: two real spatial data sets LB with $53K$ points and CA with $62K$ points presenting locations in the Long Beach country and California respectively. Data domain along each dimension is $[0, 10000]$. Given a data point p, an uncertain data U is generated with uncertainty region $U.ur$ which is a circle centering at p and has radius rad_U 250. The PDF of an uncertain object U follows either *uniform* or *Constrained-Gaussian* (Con-Gau for short) distribution. In *uniform* distribution, the distribution of an object is uniformly inside the uncertain region $U.ur$ with the same probability. The Con-Gau is based on the traditional Gaussian distribution except that the distribution of an object U is in $U.ur$.

The query region r_q is a square/cube with radius rad_q, and the distribution of its center follows that of the underlying data. In our experiments, parameter p_q (the probabilistic threshold) is ranged from 0.1 to 0.9, and rad_q (radius of query region) is from 500 to 1500. For each parameter group, 5000 queries are executed and the average result is used to measure the performance.

dimensionality d	*2*
data domain	[0, 10000] in each dimension
number of objects	53k, 62k
rad_U	250
$PDFs$	*uniform*, *Constrained-Gaussian*
number of queries	5000
rad_q	500, 1000, 1500
p_q	in [0.1, ..., 0.9]

Table 4.1: Parameter values.

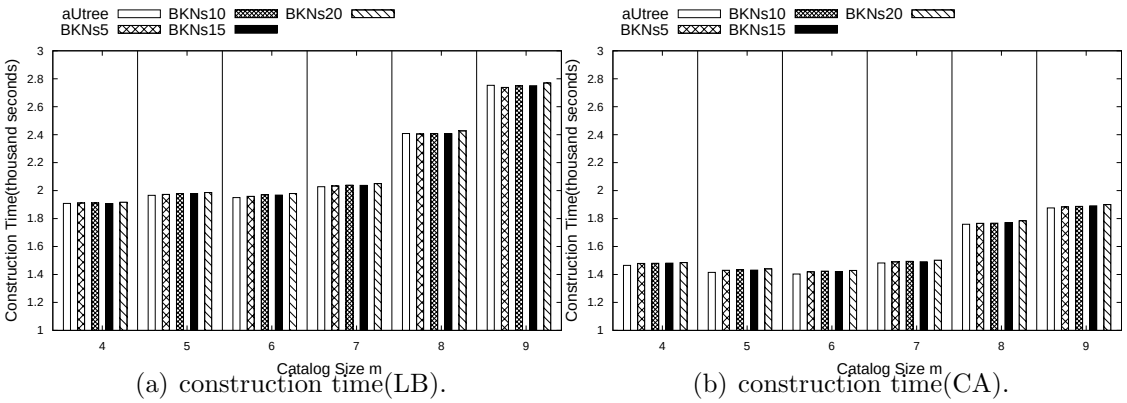
4.3.1 Construction Consumption

In practice, to construct an asU-tree, an aU-tree should be built first, then the $BKNs$ are decided on the aU-tree by Min-Skew partitioning skill, once the $BKNs$ ready, DIS is applied on each BKN to keep S samples for its Sampling Node, therefore it is inevitable that construction of asU-tree takes more time and space than aU-tree, but according to our experiment, the extra cost for asU-tree is quite limited. Figure 4.5 compares the construction consumption for aU-tree and asU-tree.

In this part of the experiment, for each data set, we set 5 groups on the number of $BKNs$ ranged from 5 to 20 and each BKN keeps 150 samples. As illustrated in Figure 4.5, the consumption to construct an asU-tree mainly lies on the procedure of building the aU-tree part. the cost for Min-Skew partitioning and DIS to form the second framework on aU-tree is almost negligible compared with inserting objects to build the tree structure as the basic framework to index uncertain data.

It should be noticed that the construction time does not go with the number of $BKNs$ in linear relationship, because besides the consumption of building tree structure, the cost is not only related with the number of $BKNs$ and the number of samples for each BKN, but also affected by the distribution of nodes in the asU-tree. If there are K nodes which can be used to separate the aU-tree into K

non-overlapping subtrees, it will save lots of time to apply Min-Skew Partitioning to re-group the aU-tree nodes into buckets.



(a) construction time(LB). (b) construction time(CA).

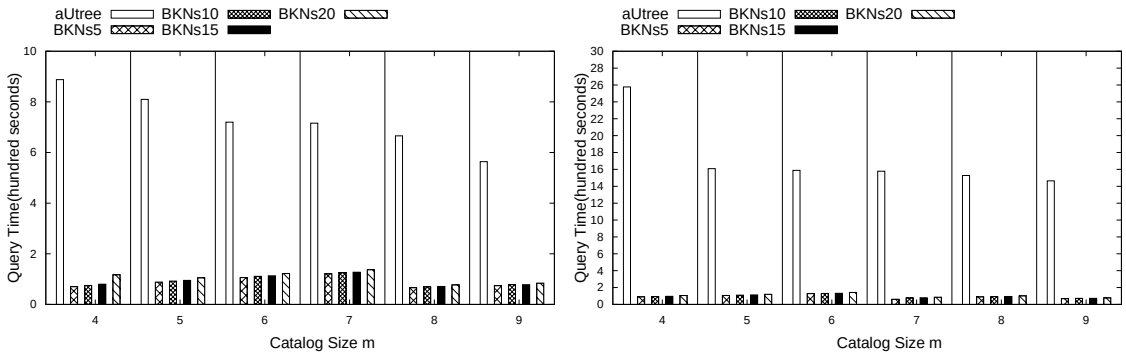
Figure 4.5: Performance vs Construction Time.

4.3.2 Efficiency Evaluation

Processing time with respect to data sets, dimensionality, query region, and probabilistic threshold is computed by a workload of 5000 queries. We evaluate the efficiency by the average time cost for all the queries.

Improving the efficiency to handle $PTRA$ queries is the major contribution of asU-tree for the $PTRA$ queries. In our experiment, we measured the efficiency of approximate query processing on asU-trees with $BKNs$ from 5 to 20 and 150 samples in each BKN. We compared to the efficiency of approximate query processing with the one of accurate query processing.

Figure 4.6 shows that for approximate query processing, if the asU-tree consists of less $BKNs$, higher efficiency will be gained, because less $BKNs$ means less nodes access to get the approximate answer. Apparently, the cost is much higher for accurate query processing.



(a) Efficiency(LB). (b) Efficiency(CA).

Figure 4.6: Efficiency Evaluation.

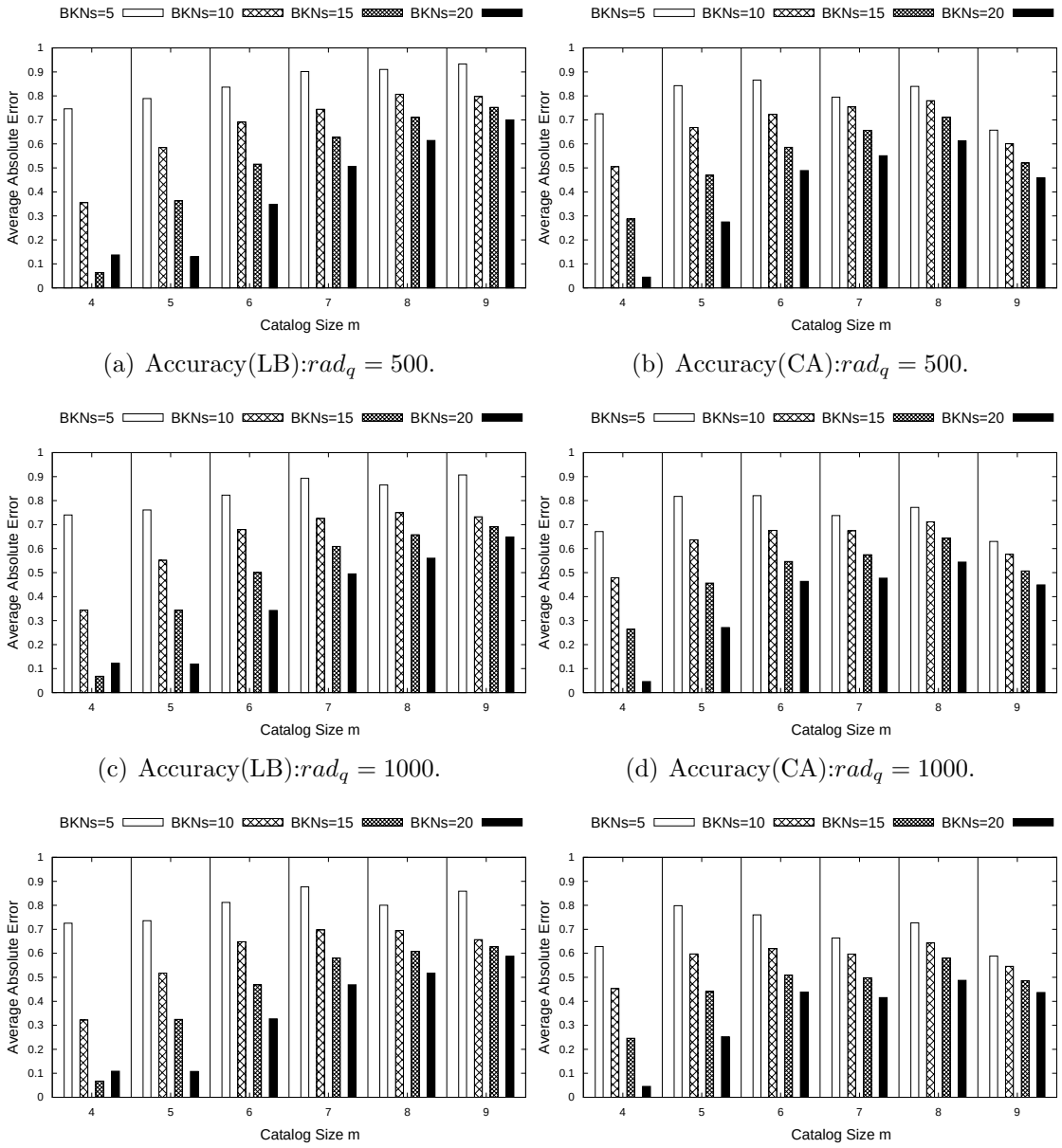
4.3.3 Accuracy Evaluation

Accuracy can be defined as a relative error of approximate answer and exact answer by the formula: $accuracy = |\frac{R_{ex}-R_{ap}}{R_{ex}}|$, where R_{ex} and R_{ap} present the exact result and approximate result, respectively.

Accuracy with respect to data sets, dimensionality, query edge length, and probabilistic threshold is also measured by a workload of 5000 queries. In this part of empirical study, the number of $BKNs$ and the number of samples are fixed the same as the efficiency evaluation part.

As illustrated in Figure 4.7, with appropriate number of $BKNs$, the accuracy can be guaranteed: the best result we can see is that the average error is less than 0.1, and most of the average errors are lower than 0.4. In the worst case, the average error reaches 0.8. If we adjust the number of $BKNs$, the approximate result could obtain high accuracy.

There are many factors working together to lead different precision besides the length of query range as illustrated in Figure 4.7, so the error has some irregularities. We are analyzing different roles of parameters on accuracy in the next section.



(a) Accuracy(LB):$rad_q = 500$. (b) Accuracy(CA):$rad_q = 500$.

(c) Accuracy(LB):$rad_q = 1000$. (d) Accuracy(CA):$rad_q = 1000$.

(e) Accuracy(LB):$rad_q = 1500$. (f) Accuracy(CA):$rad_q = 1500$.

Figure 4.7: Accuracy Evaluation.

4.3.4 Impact of the parameters

There are various factors affecting the result of the performance for both efficiency and accuracy. Our *asU*-tree is built on the basic structure of U-tree, so besides the parameters which have been revealed in [TCX+05] to show their significant

influences on the performance of asU-tree, such as the catalog size, the size of query region and the probabilistic threshold, there are two other key factors of the asU-tree which are critical for the result of the experiments: the number of the $BKNs$ and the number of samples for each BKN.

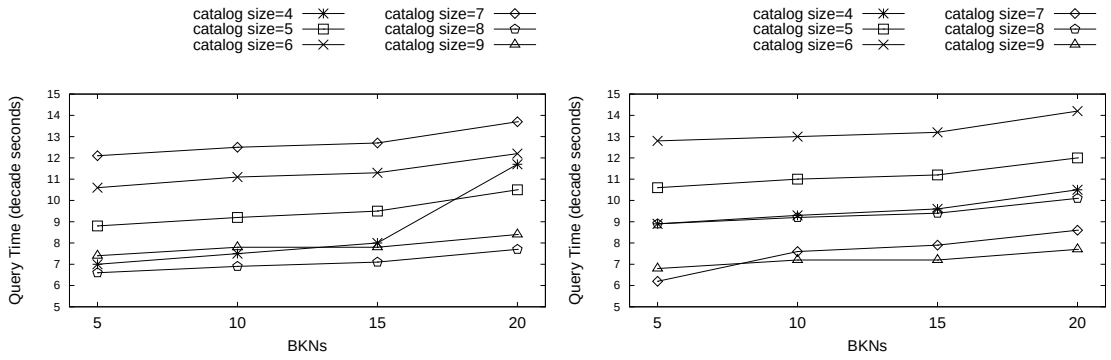
The effect of the number of the $BKNs$ on efficiency and precision

As illustrated in our experiment, the number of the $BKNs$ and the distribution of nodes are crucial for the performance of approximate query processing over asU-tree. As pointed out before, picking up the $BKNs$ is actually using Min-Skew partitioning technique on the structure of asU-tree to split the input space into K buckets. Intuitively, if raise the number of $BKNs$, the efficiency to process $PTRA$ query should be decreased because more nodes need to be retrieved, while the accuracy of the result should be improved correspondingly. This intuition is approved by the result of our experiment.

In this part of experiment, the impact of sample size is not considered, so we fix it at 150 as before and test the influence of $BKNs$ number from 5 to 20.

From Figure 4.8, it is quite obvious that between the asU-trees with different $BKNs$, the efficiency decreases when the number of $BKNs$ increases. This trend is very clear for the asU-tree for both LB data set and CA data set in Figure 4.8.

For the accuracy analysis, we can find in Figure 4.9 that the accuracy is improved with larger query region, and generally, the error decreases if the number of $BKNs$ increases, except some abnormal results revealing that the accuracy is not only affected by the number of $BKNs$, but also by the distribution of nodes on asU-tree. In Figure 4.9, for CA data set, the asU-tree with U-catalog size 4 has highest precision on 15 $BKNs$ asU-tree than 20 $BKNs$ asU-tree.



(a) LB. (b) CA.

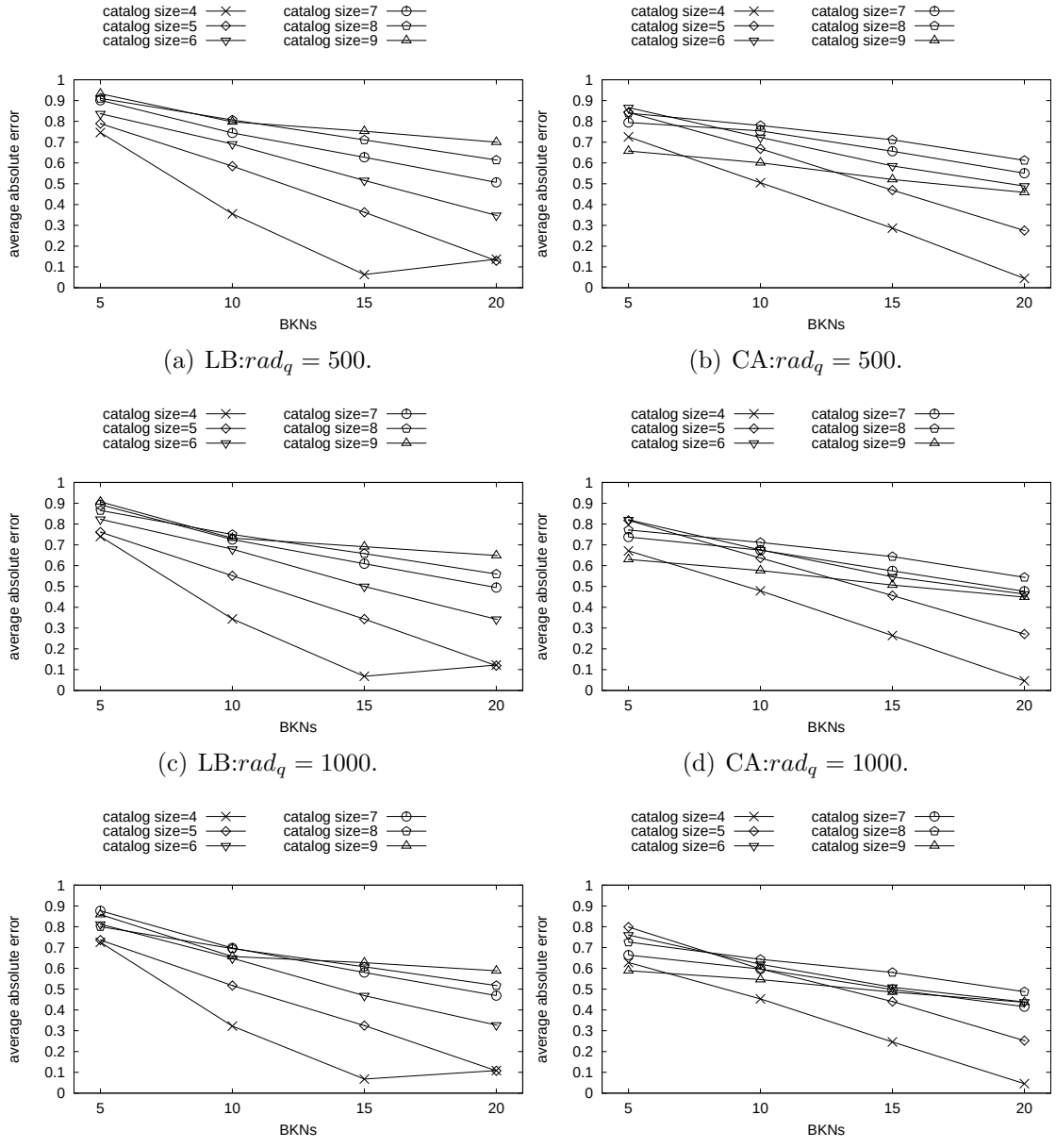
Figure 4.8: The Effect of $KBNs$ on Efficiency.

The effect of sample size on efficiency and precision

Theoretically, if there are more samples in each BKN, the approximate query processing has lower speed but higher accuracy. In this part of experiment, we analyze the effect of sample size on the result for both efficiency and accuracy.

As shown in 4.10, the procedure of $PTRA$ query performs slower as the samples increase, and the speed varies with the number of samples in linear time.

The Figure 4.11, Figure 4.12 and Figure 4.13 reveal that the accuracy of approximate result is not totally determined by the number of samples. on the other hand, the influence of the sample size is trivial. The average error does reduce while the samples increase, but the improvement is quite small.

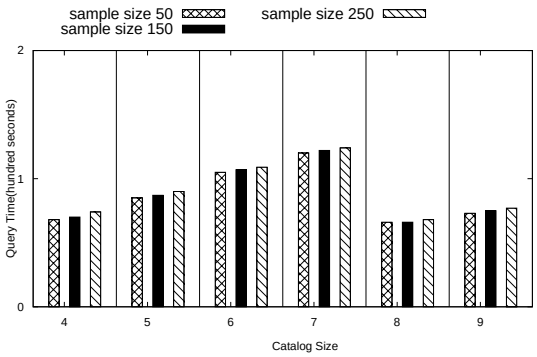


(a) LB:$rad_q = 500$. (b) CA:$rad_q = 500$.

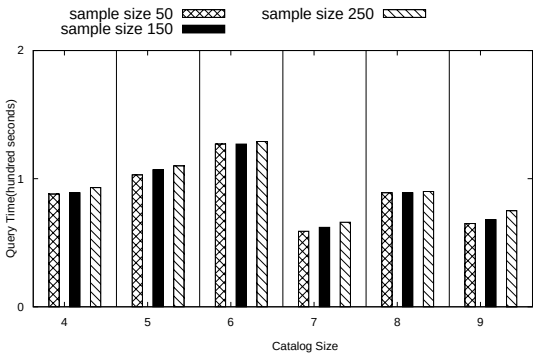
(c) LB:$rad_q = 1000$. (d) CA:$rad_q = 1000$.

(e) LB:$rad_q = 1500$. (f) CA:$rad_q = 1500$.

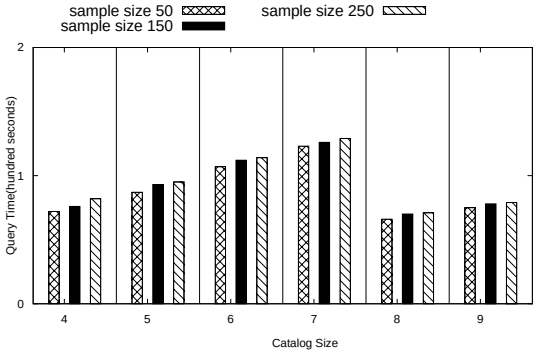
Figure 4.9: The Effect of $KBNs$ on Accuracy.



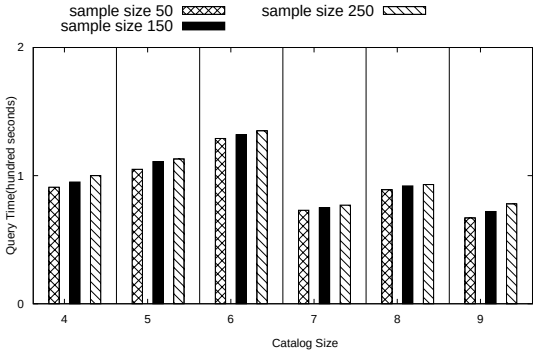
(a) 5 $BKNs$(LB).



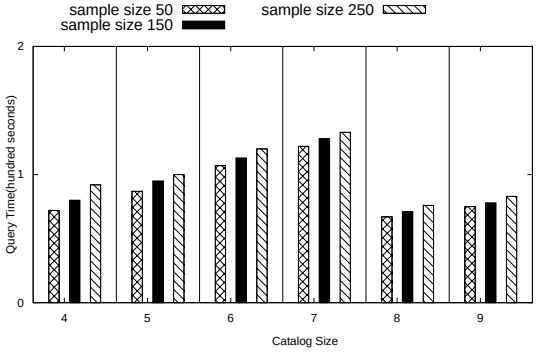
(b) 5 $BKNs$(CA).



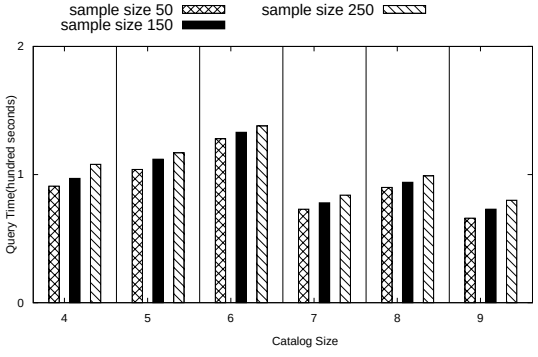
(c) 10 $BKNs$(LB).



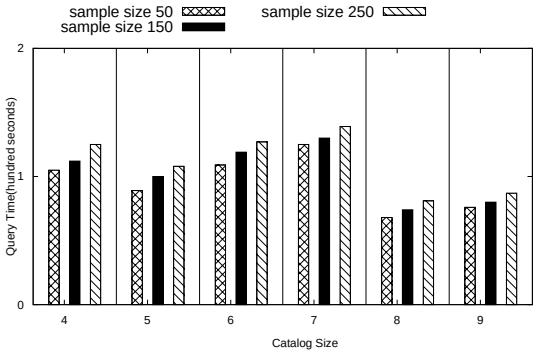
(d) 10 $BKNs$(CA).



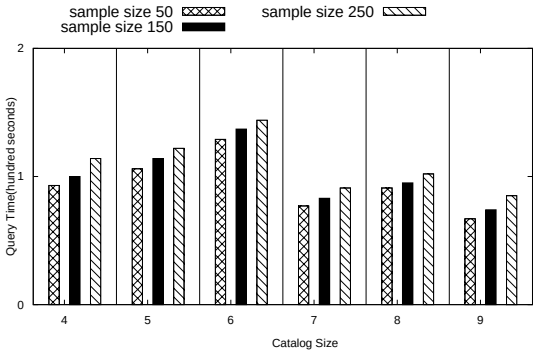
(e) 15 $BKNs$(LB).



(f) 15 $BKNs$(CA).

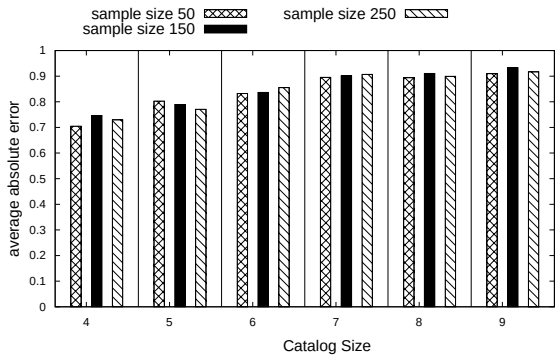


(g) 20 $BKNs$(LB).

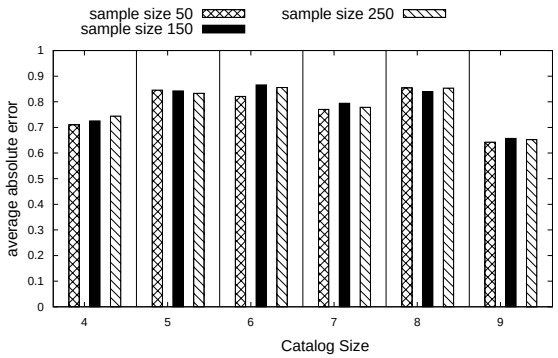


(h) 20 $BKNs$(CA).

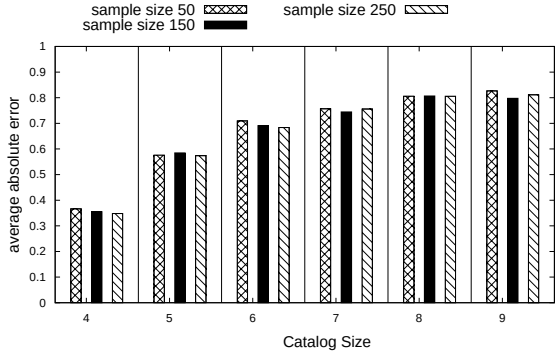
Figure 4.10: The effect of sample size on Efficiency.



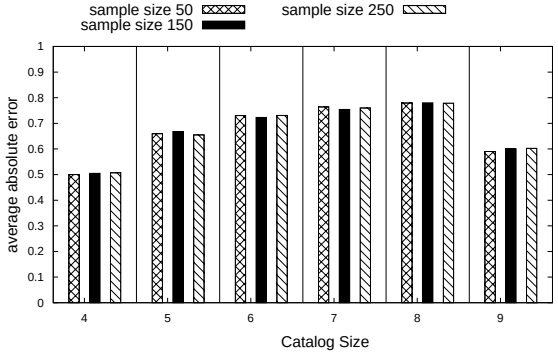
(a) 5 $BKNs$(LB):$rad_q = 500$.



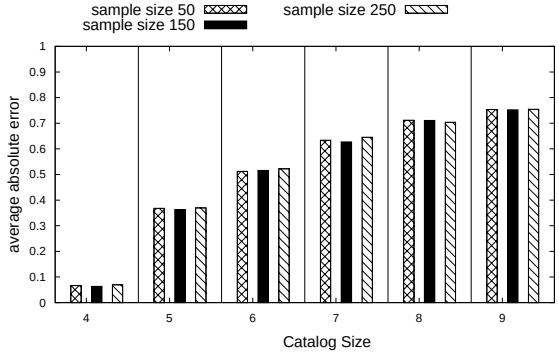
(b) 5 $BKNs$(CA):$rad_q = 500$.



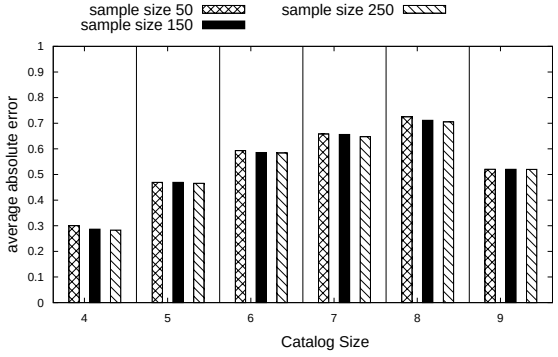
(c) 10 $BKNs$(LB):$rad_q = 500$.



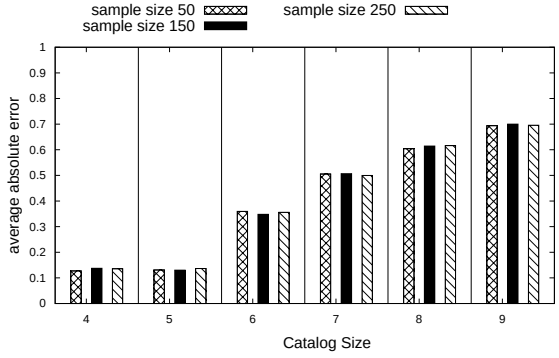
(d) 10 $BKNs$(CA):$rad_q = 500$.



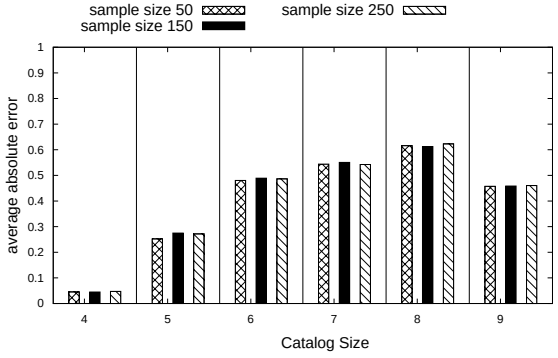
(e) 15 $BKNs$(LB):$rad_q = 500$.



(f) 15 $BKNs$(CA):$rad_q = 500$.

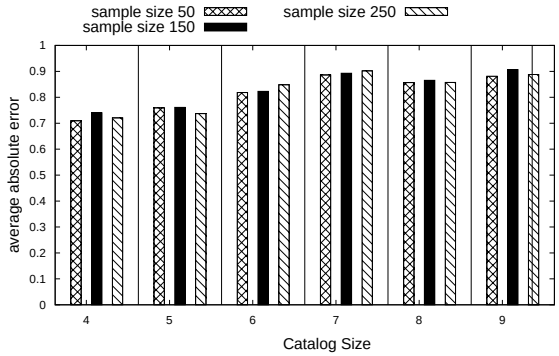


(g) 20 $BKNs$(LB):$rad_q = 500$.

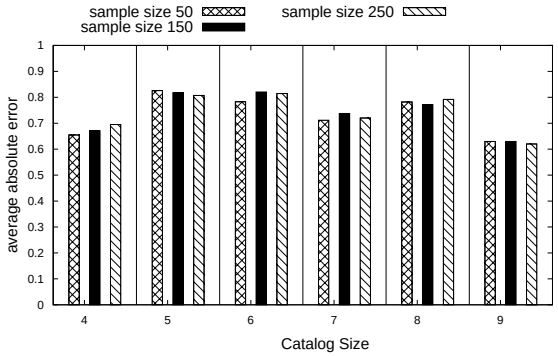


(h) 20 $BKNs$(CA):$rad_q = 500$.

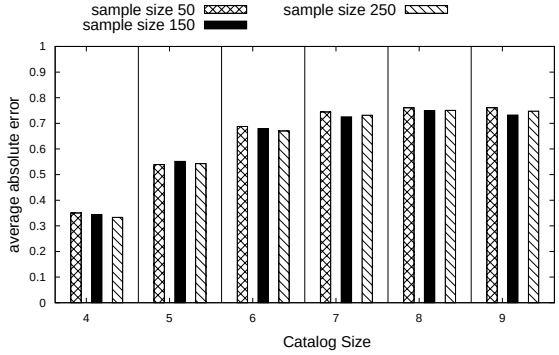
Figure 4.11: The effect of sample size on Accuracy($rad_q = 500$).



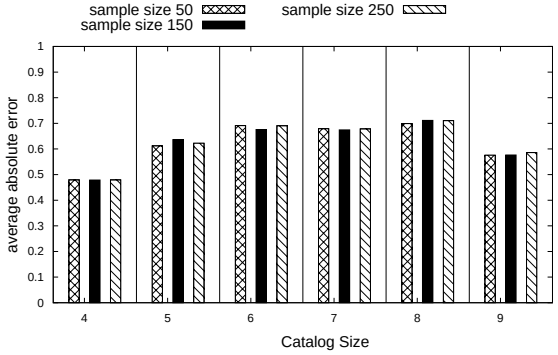
(a) 5 $BKNs$(LB):$rad_q = 1000$.



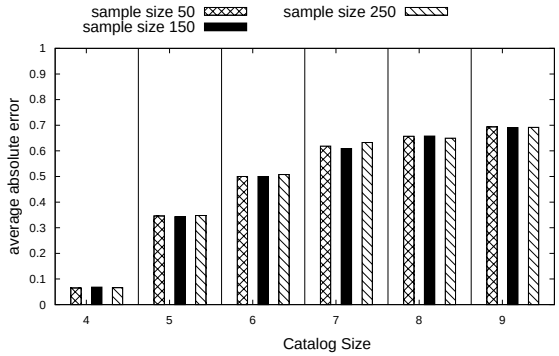
(b) 5 $BKNs$(CA):$rad_q = 1000$.



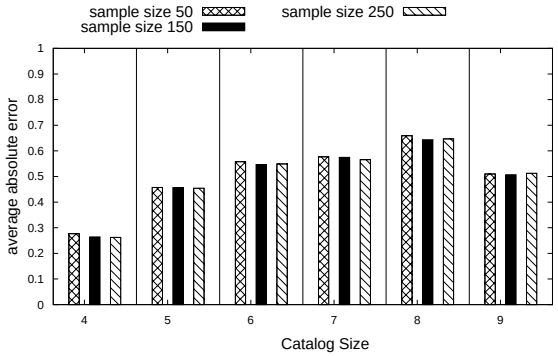
(c) 10 $BKNs$(LB):$rad_q = 1000$.



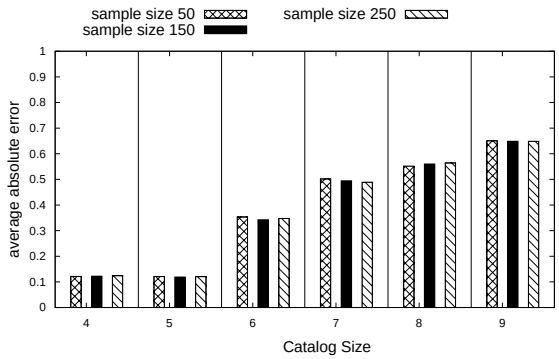
(d) 10 $BKNs$(CA):$rad_q = 1000$.



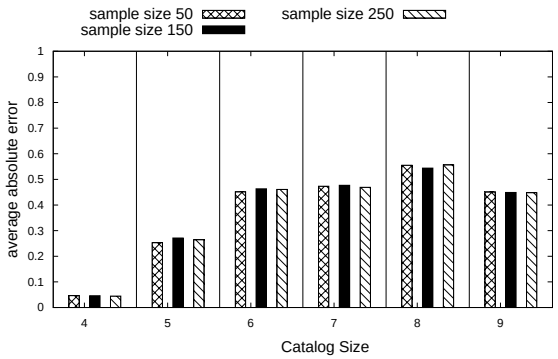
(e) 15 $BKNs$(LB):$rad_q = 1000$.



(f) 15 $BKNs$(CA):$rad_q = 1000$.

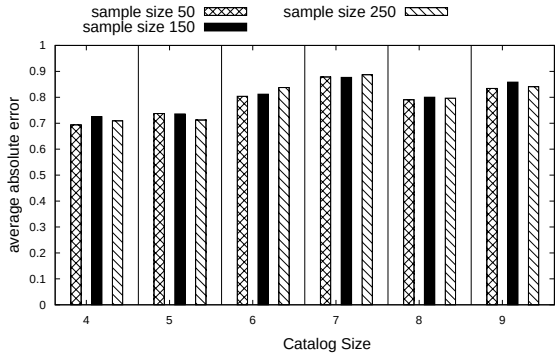


(g) 20 $BKNs$(LB):$rad_q = 1000$.

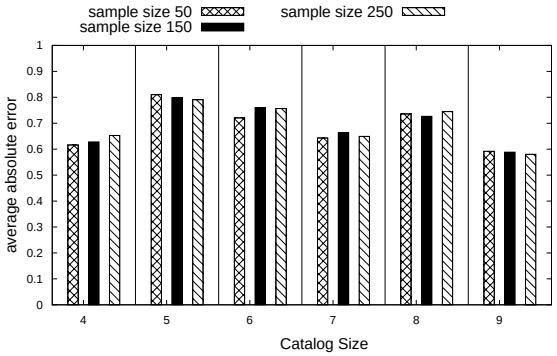


(h) 20 $BKNs$(CA):$rad_q = 1000$.

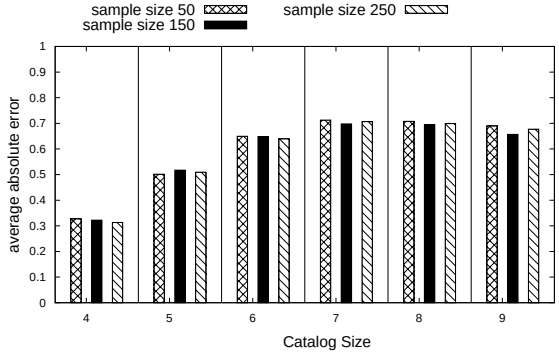
Figure 4.12: The effect of sample size on Accuracy($rad_q = 1000$).



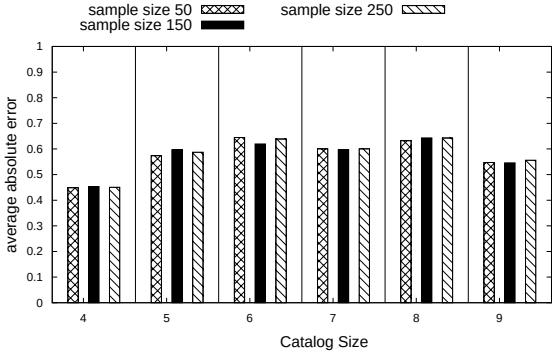
(a) 5 $BKNs$(LB):$rad_q = 1500$.



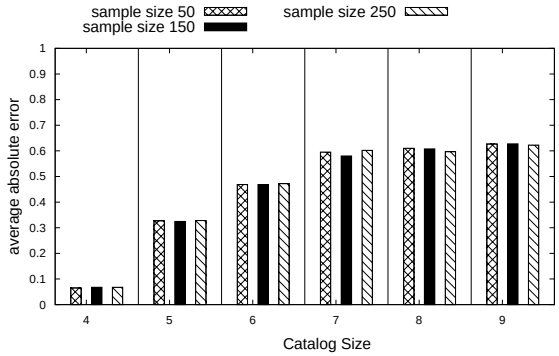
(b) 5 $BKNs$(CA):$rad_q = 1500$.



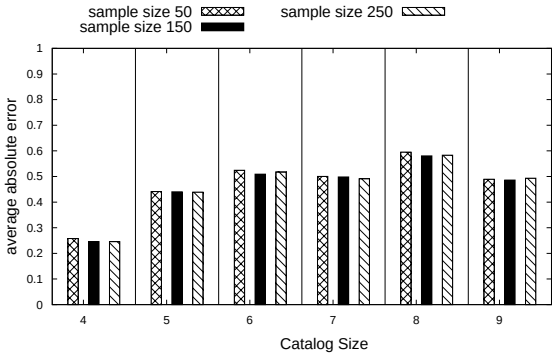
(c) 10 $BKNs$(LB):$rad_q = 1500$.



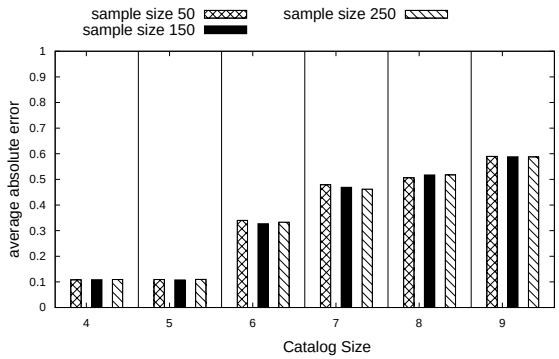
(d) 10 $BKNs$(CA):$rad_q = 1500$.



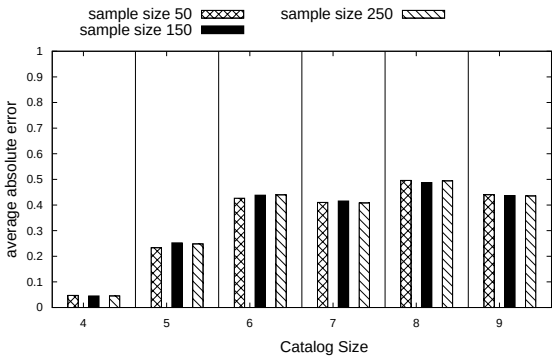
(e) 15 $BKNs$(LB):$rad_q = 1500$.



(f) 15 $BKNs$(CA):$rad_q = 1500$.



(g) 20 $BKNs$(LB):$rad_q = 1500$.



(h) 20 $BKNs$(CA):$rad_q = 1500$.

Figure 4.13: The effect of sample size on Accuracy($rad_q = 1500$).

Chapter 5

Conclusion and Future Work

An important problem, probabilistic threshold range aggregate query is investigated in this thesis. After formally defining this problem, we firstly propose a novel index structure, asU-tree, which integrates aggregate information for range aggregate query as well as sampling techniques to support approximate query. Exact and approximate query processing algorithms are developed based on the asU-tree. Our comprehensive experimental study confirms the efficiency and effectiveness of the techniques we proposed.

As a future work direction, firstly, we will optimize the algorithm to split the asU-tree into K subtrees. Min-Skew partitioning works very well on spatial space, but in our case, Min-Skew partitioning is applied on the aU-tree which is of hierarchical structure. If the insertion or deletion leads to modification on the nodes distribution over asU-tree, the $BKNs$ need to be adjusted by applying Min-Skew partitioning all over again.

Secondly, we are trying to find an approach to choose the best number of K for different asU-tree. From the experiment, the performance is affected by both the number of $BKNs$ and the nodes distribution of asU-tree. When the number of

$BKNs$ brings less splitting by Min-Skew partitioning, the result has more efficiency and accuracy.

Finally, we will study the performance of other summarizing techniques, such as histograms [TGIK02], to provide approximate answers for range aggregate queries over uncertain data, and make the summarizing technique works well with the index technique.

Bibliography

[ABS+06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha U. Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *VLDB*, pages 1151–1154, 2006.

[ACc+03] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, C. Erwin, Eduardo F. Galvez, M. Hatoun, Anurag Maskey, Alex Rasin, A. Singer, Michael Stonebraker, Nesime Tatbul, Ying Xing, R. Yan, and Stanley B. Zdonik. Aurora: A data stream management system. In *SIGMOD Conference*, page 666, 2003.

[APR99] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity estimation in spatial databases. In *SIGMOD Conference*, pages 13–24, 1999.

[AW] P. Agrawal and J. Widom. Confidence-aware joins in large uncertain databases. In *Stanford University Technical Report, 2007.*

[AY08] Charu C. Aggarwal and Philip S. Yu. On high dimensional indexing of uncertain data. In *ICDE*, pages 1460–1461, 2008.

[BGK+07] Christian Böhm, Michael Gruber, Peter Kunath, Alexey Pryakhin, and Matthias Schubert. Prover: Probabilistic video retrieval using the gauss-tree. In *ICDE*, pages 1521–1522, 2007.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 322–331, 1990.

[BPS06a] Christian Böhm, Alexey Pryakhin, and Matthias Schubert. The gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *ICDE*, page 9, 2006.

[BPS06b] Christian Böhm, Alexey Pryakhin, and Matthias Schubert. Probabilistic ranking queries on gaussians. In *SSDBM*, pages 169–178, 2006.

[CCD+03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Frederick Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing. In *SIGMOD Conference*, page 668, 2003.

[CCKN06] Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng. Uncertain data mining: An example in clustering location data. In *PAKDD*, pages 199–204, 2006.

[CCMC08] Reynold Cheng, Jinchuan Chen, Mohamed F. Mokbel, and Chi-Yin Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *ICDE*, pages 973–982, 2008.

[CCT96] Arbee L. P. Chen, Jui-Shang Chiu, and Frank Shou-Cheng Tseng. Evaluating aggregate operations over imprecise data. volume 8, pages 273–284, 1996.

[CJSS03] Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *SIGMOD Conference*, pages 647–651, 2003.

[CKP03] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 551–562, 2003.

[CKP07] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluation of probabilistic queries over imprecise data in constantly-evolving environments. *Information Systems*, 32(1):104–130, 2007.

[CM05] Graham Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. volume 30, pages 249–278, 2005.

[CMN98] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. Random sampling for histogram construction: How much is enough? In *SIGMOD Conference*, pages 436–447, 1998.

[CMN99] Surajit Chaudhuri, Rajeev Motwani, and Vivek R. Narasayya. On random sampling over joins. In *SIGMOD Conference*, pages 263–274, 1999.

[CMR05] Graham Cormode, S. Muthukrishnan, and Irina Rozenbaum. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In *VLDB*, pages 25–36, 2005.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems (TODS)*, 4(4):397–434, 1979.

[CSP+06] Reynold Cheng, Sarvjeet Singh, Sunil Prabhakar, Rahul Shah, Jeffrey Scott Vitter, and Yuni Xia. Efficient join processing over uncertain data. In *CIKM*, pages 738–747, 2006.

[CXP+04] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffrey Scott Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of 30th International Conference on Very Large Data Bases (VLDB)*, pages 876–887, 2004.

[DS04] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *Proceedings of 30th International Conference on Very Large Data Bases (VLDB)*, pages 864–875, 2004.

[DS05] Nilesh N. Dalvi and Dan Suciu. Answering queries from statistics and probabilistic views. In *Proceedings of 31st International Conference on Very Large Data Bases (VLDB)*, pages 805–816, 2005.

[DS07] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS)*, pages 1–12, 2007.

[FR97] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Transactions on Information Systems (TOIS)*, 15(1):32–66, 1997.

[Fuh95] Norbert Fuhr. Probabilistic datalog - a logic for powerful retrieval methods. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR)*, pages 282–290, 1995.

[GM98] Phillip B. Gibbons and Yossi Matias. New sampling-based summary statistics for improving approximate query answers. In *SIGMOD Conference*, pages 331–342, 1998.

[GUP06] Jose Galindo, Angelica Urrutia, and Mario Piattini. *Fuzzy Databases: Modeling, Design, and Implementation*. Idea Group Publishing, 2006.

[Gut84] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 47–57, 1984.

[HPZL08a] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *ICDE*, pages 1403–1405, 2008.

[HPZL08b] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Ranking queries on uncertain data: a probabilistic threshold approach. In *SIGMOD Conference*, pages 673–686, 2008.

[IJ84] Tomasz Imielinski and Witold Lipski Jr. Incomplete information in relational databases. *Journal of the ACM*, 31(4):761–791, 1984.

[INV91] Tomasz Imielinski, Shamim A. Naqvi, and Kumar V. Vadaparty. Incomplete objects - a data model for design and planning applications. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 288–297, 1991.

[JKV07] T. S. Jayram, Satyen Kale, and Erik Vee. Efficient aggregation algorithms for probabilistic data. In *SODA*, pages 346–355, 2007.

[JL] M. Jurgens and H. Lenz. The ra*-tree: An improved r-tree with materialized data for supporting range queries on olap-data. In *DEXA workshop, 1998*.

[JPY07] Xuemin Lin Jian Pei, Bin Jiang and Yidong Yuan. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, 2007.

[KKR07] Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *DASFAA*, pages 337–348, 2007.

[KP05] Hans-Peter Kriegel and Martin Pfeifle. Density-based clustering of uncertain data. In *KDD*, pages 672–677, 2005.

[L08] Xiang Lian and Lei Chen 0002. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *SIGMOD Conference*, pages 213–226, 2008.

[Lee92] Suk Kyoon Lee. An extended relational database model for uncertain and imprecise information. In *Proceedings of 18th International Conference on Very Large Data Bases (VLDB)*, pages 211–220, 1992.

[LM01] Iosif Lazaridis and Sharad Mehrotra. Progressive approximate aggregate queries with a multi-resolution tree structure. In *SIGMOD Conference*, pages 401–412, 2001.

[LS07] Vebjorn Ljosa and Ambuj K. Singh. Apla: Indexing arbitrary probability distributions. In *ICDE*, pages 946–955, 2007.

[MM02] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of 28th International Conference on Very Large Data Bases (VLDB)*, pages 346–357, 2002.

[MSS01] Sally I. McClean, Bryan W. Scotney, and Mary Shapcott. Aggregation of imprecise and uncertain information in databases. volume 13, pages 902–912, 2001.

[MW] R. Murthy and J. Widom. Making aggregation work in uncertain and probabilistic databases. In *MUD workshop, 2007.*

[NKC+06] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip. Efficient clustering of uncertain data. In *ICDM*, pages 436–445, 2006.

[Olk] F. Olken. Random sampling from databases. In *PhD thesis, Berkeley.*

[PKZT01] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *SSTD*, pages 443–459, 2001.

[PTVF] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. Numerical recipes in c++. In *Cambridge University Press, 2002.*

[RB91] Elke A. Rundensteiner and Lubomir Bic. Evaluating aggregates in possibilistic relational databases. volume 7, pages 239–267, 1991.

[RDS] C. Re, N. Dalvi, and D. Suciu. Efficient top-k query evaluation on probabilistic data. In *Proceedings of the 23th International Conference on Data Engineering(ICDE), 2007.*

[RSG05] Robert B. Ross, V. S. Subrahmanian, and John Grant. Aggregate operators in probabilistic databases. volume 52, pages 54–101, 2005.

[SBHW06] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, and Jennifer Widom. Working models for uncertain data. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE)*, page 7, 2006.

[SD07] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *ICDE*, pages 596–605, 2007.

[SIC] M. A. Soliman, I. F. Ilyas, and K. C. Chang. Top-k query processing in uncertain databases. In *Proceedings of the 23th International Conference on Data Engineering(ICDE), 2007.*

[SM] B. Scotney and S. McClean. Database aggregation of imprecise and uncertain evidence. In *Inf. Sci. Inf. Comput. Sci, 155(3):245C263, 2003.*

[SMP+07] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne E. Hambrusch. Indexing uncertain categorical data. In *ICDE*, pages 616–625, 2007.

[TCX+05] Yufei Tao, Reynold Cheng, Xiaokui Xiao, Wang Kay Ngai, Ben Kao, and Sunil Prabhakar. Indexing multi-dimensional uncertain data with

arbitrary probability density functions. In *Proceedings of 31st International Conference on Very Large Data Bases (VLDB)*, pages 922–933, 2005.

[TGIK02] Nitin Thaper, Sudipto Guha, Piotr Indyk, and Nick Koudas. Dynamic multidimensional histograms. In *SIGMOD Conference*, pages 428–439, 2002.

[TP04] Y. Tao and D. Papadias. Range aggregate processing in spatial databases. *ACM TODS*, 16(12):1555–1570, 2004.

[Vit] J. S. Vitter. Random sampling with a reservoir. In *ACM Transactions on Mathematical Software, 11(1):37C57, 1985.*

[YLSK] K. Yi, F. Li, D. Srivastava, and G. Kollios. Efficient processing of top-k queries in uncertain databases. In *Technical report, Florida State University, 2007.*

[ZLPZ08] Wenjie Zhang, Xuemin Lin, Jian Pei, and Ying Zhang. Managing uncertain data: Probabilistic approaches. In *WAIM*, pages 405–412, 2008.

[ZLY08] Qin Zhang, Feifei Li, and Ke Yi. Finding frequent items in probabilistic data. In *SIGMOD Conference*, pages 819–832, 2008.