Efficiently and Effectively Processing Probabilistic Queries on Uncertain Data

by

Wenjie Zhang

B.Sc., Harbin Institute of Technology, 2004

M.Sc., Harbin Institute of Technology, 2006

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

IN THE SCHOOL

OF

Computer Science and Engineering

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY · AUSTRALIA

Aug 6, 2010

Originality Statement

Abstract

Uncertainty is inherent in data collected from many important, novel applications such as large sensor networks, WWW, data cleaning and integration, environmental surveillance and market analysis. The sources of uncertainty in these applications vary from data randomness and incompleteness, limitations of measuring equipments, delay or loss in data transfer. As a rapidly growing amount of uncertain data is collected, it is highly desirable to conduct advanced analyzing and query processing over uncertain data. The following five important aspects for uncertain data management are investigated in this thesis.

We study the problem of *probabilistic top-k skyline queries*. A model for the top-k skyline operator is proposed combining the feature of top-k objects and that of skyline. Based on this model, an efficient exact algorithm and a randomized algorithm with ϵ-approximation guarantee are developed for discrete and continuous cases, respectively.

We extend skyline operator to streaming environment and study the problem of *probabilistic skyline queries over sliding windows*. We characterize the minimum information needed in continuously computing probabilistic skyline against a sliding window. Then novel, efficient techniques are developed to process a continuous, probabilistic skyline query.

As the top-k dominating query is another important method for multi-criterion decision making, we study *top-k dominating queries on uncertain data.* The problem is formally defined in a probability threshold fashion. Then, a threshold-based algorithm is developed to compute the exact solution. To overcome some inherent computational deficiency in an exact computation, we develop an efficient randomized algorithm with an accuracy guarantee.

We study the problem of *quantile-based KNN over multi-valued objects.* Two different ϕ-quantile distances are proposed. While the first distance can be computed in polynomial time, the second problem is NP-hard. A set of efficient, novel algorithms have been proposed to give an exact solution for the first problem and an approximate solution for the second problem with the approximation ratio 2.

To overcome some deficiencies of existing uncertain index structures, we propose *UI-tree* which can efficiently support various queries including range queries, similarity joins and their size estimation, as well as top-k range query, over multi-dimensional uncertain objects against continuous or discrete cases.

Dedication

Father, Yunquan Zhang

Mother, Zhenfeng Wu

Sister, Wenchao Zhang

For their love and support

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Prof. Xuemin Lin. During my PhD study, Prof. Lin encourages me to explore my potential in research and spends great efforts on the supervision. I also learnt the characteristics of a successful researcher from Prof. Lin. He has great passion for research, paying attention to details and trying best in every research project involved.

I would like to thank Prof. Jeffrey Xu Yu, Prof. Jian Pei, Dr. Wei Wang, Dr. Ying Zhang, Dr. Ming Hua and Aamir Cheema for the research collaboration in the work presented in Chapter 3, 4, 5, 6 and 7.

And special thanks go to the former group members: Yidong Yuan and Bin Jiang, as well as the fellow group members: Yi Luo, Mahady Hansan, Chuan Xiao, Haichuan Shang, Gaoping Zhu, Ke Zhu, Zhitao Shen, Weiren Yu, Weiwei Jia, Pengjie Ye, Xiang Zhao, Liming Zhan and Jing Yang. The time we spent together will be memorized forever.

I would like to thank Prof. Xiaofang Zhou in University of Queensland and Prof. Jianmin Wang in Tsinghua University for providing me internship opportunities in their research groups during my PhD life.

I am grateful to the staff members from our school. I am also very grateful to the University of New South Wales and NICTA for financially supporting my PhD study.

Last but not least, I would also like to thank my parents and my younger sister for their love and support.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Data stored in traditional databases are required to be modeled as precise values. In recent years, many emerging, important applications produce a large amount of uncertain data. These applications include sensor networks, trend prediction, moving object management, data cleaning and integration, economic decision making, market surveillance and privacy preservation. In such applications, uncertainty is inherent due to various factors such as data randomness, limitation of equipment, and delay or loss in data transfer. The uncertain nature poses great challenges for uncertain data management. For instance, the problem of evaluating conjunctive queries can be #P-complete over uncertain data [DS07a]. Thus, many query types have been reinvestigated under the uncertain semantics, including query evaluation [DS04], aggregate queries [BDRV05], joins [CSP06], top-k queries [HPZL08b], skyline queries [PJLY07], dominating queries [ZLZ+09], nearest neighbor queries [BSI08], etc.

1.1 Applications

We enumerate some typical applications in this section.

Sensor Networks Wireless sensor networks (WSNs) are widely deployed with a set of sensor nodes installed in a large range. Certain tasks will be carried out by the sensor nodes such as monitoring temperature, humility and wind speed in a forest to decrease the risk of forest fires. In many cases WSNs are set up in remote areas or areas that are difficult for humans to access; also the sensor nodes are often battery-powered. Thus, limited energy of the sensor nodes is a key challenge in WSNs management. For the sake of energy conservation, according to some protocols, the sensor nodes usually follow the hibernate–wakeup life circle which means they hibernate for some time and then wake up collecting data and send readings to the server. So the readings kept at the server are not always precise but associated with certain levels of uncertainty. Another source of uncertainty in WSNs origins from the limitation of equipment which means the readings of a sensor node are inherently imprecise or erroneous. In order to detect areas in a forest with high risks of fire hazard, people may want to select areas with high temperature, low humility and high wind speed from all areas with sensor nodes installed. Such kind of multi-criteria decision making problems are often modeled as skyline or dominating queries and should be conducted on the collected uncertain data.

As a concrete example, suppose sensors are used to detect the temperature and wind speed in a forrest. Due to limitations of sensors, detections can not be accurate all the time. Instead, detection confidence is often estimated. Table 1.1 lists a set of synthesized records of parameters detected by sensors. In some high harzard locations, multiple sensors are deployed to improve the detection quality. Two sensors in the same location (e.g., S206 and S231, as well as S063 and S732 in Table 1.1) may detect the parameters at (approximately) the same time, such as records R2 and R3, as well as R5 and R6. In such a case, if the readings detected by multiple sensors are inconsistent, at most one sensor can be correct.

RID	**Location**	**Time**	**Sensor-id**	**Temp.**	**Wind-Speed**	**Conf.**
$R1$	A	2:14 AM	S101	21 °C	20 km/h	0.3
$R2$	B	12:07 PM	S206	33 °C	51 km/h	0.4
$R3$	B	12:09 PM	S231	35 °C	55 km/h	0.5
$R4$	A	1:32 PM	S101	34 °C	48 km/h	1.0
$R5$	E	2:31 PM	S063	37 °C	57 km/h	0.8
$R6$	E	2:28 PM	S372	40 °C	56 km/h	0.2

Table 1.1: Readings of Temperature and Wind-Speed

Location Based Services With the rapid development of wireless communication and GPS techniques, there is a huge amount of location information of moving objects collected and accumulated everyday. Usually the GPS equipped moving objects report their location information to a server through wireless communication networks. However, to do the reporting continuously consumes too much network connection and shortens the battery lifetime. Instead, the moving object may only report its location periodically. For example, the location of a set of iphone holders may be monitored. In Figure 1.1 [PHTL], an iphone reports its location to the server every five minutes. So even though we know the position of the iphone holder exactly at time stamps 10:00 AM and 10:05 AM, the position information in-between these two time stamps is not known. Instead, based on some information like road network and speed constraint, we may expect the position of the iphone holder in between 10:00AM and 10:05AM to be within the shaded area.

Image Data Analysis Orbit satellites may record sightings in a large range remotely so that they are often used to detect movement of enemies in battle fields. However, due to the limitation of sight capturing devices in satellites and the fact that a satellite orbits in the space far away, the sightings captured are often very vague. As shown in Figure 1.2 [AKO07], based on the satellites sightings, we may discover there is a moving object in the red circle; however, we can not conclude

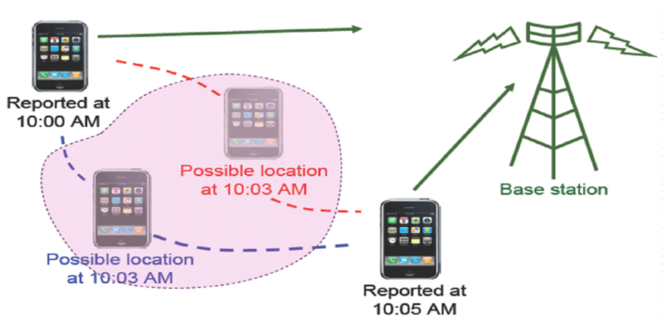
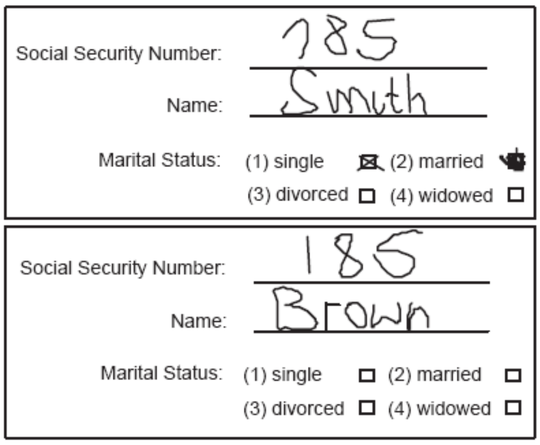


Figure 1.1: Location of iPhone Holders

whether this object is our enemy or friend and whether it is for civilian or military uses.

Figure 1.2: Sightings of a Satellite

Data Quality Another typical application involving uncertain data is data cleaning and integration. For instance, the data collected from a manually filled form for the sake of census may contain missing or unclear values, as shown in Figure 1.3 [AKO07]. In the first form the marital status is not clear and in the second, this information is completely missing. Furthermore, the social security number in the first form can be read as either “785” or “185” and we can not tell if the social security number in the second form is “186” or “185”.



Social Security Number: 785

Name: Smith

Marital Status: [✓ (1) single] [✓ (2) married]
(3) divorced ☐ (4) widowed ☐

Social Security Number: 185

Name: Brown

Marital Status: (1) single ☐ (2) married ☐
(3) divorced ☐ (4) widowed ☐

Figure 1.3: Two Survey Forms

Other applications requiring uncertain data management include social networks, information retrieval, economic decision making, etc.

1.2 Types of Uncertain Data

Note that throughout this thesis, we do not distinguish between imprecise and uncertain data and use the term uncertain for the reason of simplicity. To be precise, imprecision means information available is not specific enough, for instance, the temperature outside is between 35 and 38 centigrade (interval); it is 35 or 38 centigrade (disjunction); it is not 20 centigrade (negative); or we simply do not know the outside temperature. On the other hand, uncertainty indicates it is impossible to determine whether information available is true or not. For instance, the temperature may be 38 centigrade [GUP05].

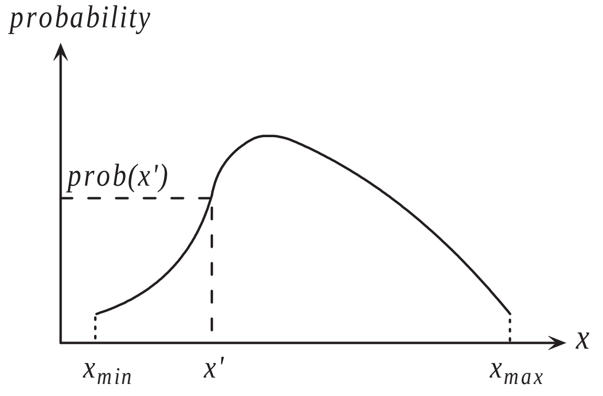
Most frequently used granularities to specify uncertain are group-based, object-based and instance-based [TCX+05]. A group-based approach concerns the "coverage" of the group such as how much percent of objects in the group is present; an object-based approach assigns appearance probability to each object in the group;

in the instance level, an instance from an uncertain object is associated with probability distribution information describing a set of possible values. In this thesis, we mainly focus on the instance-based granularity of uncertainty.

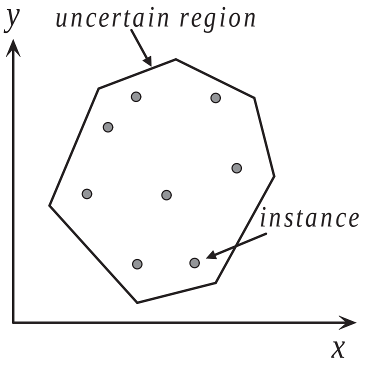
There are two major cases of uncertain data in most applications, the *continuous* case and the *discrete* case. An uncertain object U may be described by a probability density function (PDF) f_U such that $\int_{u \in U} f_U(u)du = 1$ where $f_U(u) \geq 0$; this is also referred as the continuous case. Nevertheless, in many applications PDFs are not always available. Instead, an uncertain object U is represented by a set of instances (points) such that each instance $u \in U$ has a probability p_u to appear. Such a representation, also referred as the discrete case, has the property that $0 < p_u \leq 1$ and $\sum_{u \in U} p_u = 1$.

In the following part of the thesis, in the continuous case, an uncertain object U in a d-dimensional space is defined with (i) a PDF $f_U(u)$ where u is d-dimensional point; (ii) a d-dimensional uncertain region covering all possible locations of $u \in U$. In the discrete case, U is defined with (i) a set of instances such that each instance u is a point in the d-dimensional space and $\sum_{u \in U} p_u = 1$, $0 < p_u \leq 1$; (ii) a d-dimensional uncertain region covering all the possible locations of $u \in U$.

Figure 1.4 describes two uncertain objects. Figure 1.4 (a) shows a 1-dimensional uncertain object with a continuous PDF. $[x_{min}, x_{max}]$ represents the uncertain region of the object and for any $x' \in [x_{min}, x_{max}]$, the associated probability of x' can be obtained precisely using the given PDF. Figure 1.4 (b) illustrates a 2-dimensional uncertain object in a discrete case. The irregular polygon represents the uncertain region covering locations of all instances in the object.



(a) Continuous Case



(b) Discrete Case

Figure 1.4: Continuous and Discrete Cases

1.3 Problems Studied

From the view of database research, uncertain data analysis largely contains three steps, data collection, uncertainty integration, and modeling and querying uncertain data, as shown in Figure 1.5. In data collection, large amounts of uncertain data are collected through sensor nodes, GPS equipped devices, survey conduction, etc. In uncertainty integration, the uncertainty level is assessed. Probability values may be assigned to uncertain data based on numerous factors including the nature of the equipment and confidence level of observation. Also, other information such as correlations among uncertain data are derived based on applications. In this thesis, we mainly focus on the last step, namely, modeling and querying uncertain data.

Numerous applications will benefit from the support of advanced query types and efficient indexing techniques on uncertain data. However, research on these aspects still remains largely open. Particularly, *skyline query* and *dominating query*, as two important tools in multi-criteria decision making, need to be reinvestigated under the uncertain semantics. Secondly, *KNN queries* are widely used in various

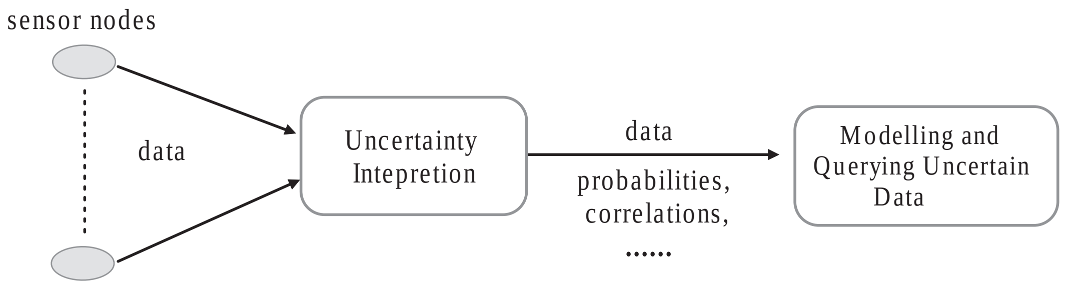

sensor nodes
data
Uncertainty Intepretion
data
probabilities, correlations, ...
Modelling and Querying Uncertain Data

Figure 1.5: Uncertain Data Analysis Framework

applications to retrieve a set of objects that are closest to a query object according to given distance metrics. According to our analysis in Chapter 6, existing techniques for KNN queries on uncertain data may not fully capture the distributions of instances. Lastly, a set of queries rely on efficient processing of range queries, including range aggregates, join, similarities joins, nearest neighbor queries, skyline queries, etc. However, current *indexing techniques* supporting range query on uncertain data are sensitive to the size or shape of uncertain regions of uncertain objects and queries.

Motivated by the above observations, this thesis aims to bridge the gap of advanced query types and efficient indexing techniques in the field of uncertain data management.

Problem I: The Probabilistic Top-k Skyline Operator

Given a set $\mathcal{U}$ of uncertain objects and an integer k, the probabilistic top-k skyline operator retrieves k uncertain objects with maximal skyline probabilities.

Problem II: Probabilistic Skyline Operator over Sliding Windows

We study the problem of efficiently retrieving skyline elements from the most recent N elements, seen so far in a stream, with the skyline probabilities not smaller than a given threshold q $(0 < q \leq 1)$; that is, q-skyline. Specifically, we will investigate the problem of efficiently processing such a continuous query, as well as ad-hoc queries with a probability threshold $q' \geq q$.

Problem III: Probabilistic Top-k Dominating Queries

Given a probabilistic threshold q, an integer k, and a set $\mathcal{U}$ of uncertain objects, a threshold based probabilistic top-k dominating query retrieves k objects with highest dominating probability. Dominating probability for an uncertain object $U \in \mathcal{U}$ is defined as the largest number of other objects in $\mathcal{U}$ that are dominated by U with probability at least q.

Problem IV: Quantile-based KNN over Multi-valued Objects

Each multi-valued object is represented by a set of instances and the sum of probabilities of the instances equals to 1. Given a value $\phi \in (0, 1]$, we define two types of distance between two multi-valued objects Q and U, ϕ-quantile distance $d_\phi(Q, U)$ and ϕ-quantile group-base distance $gbd_\phi(Q, U)$.

[**ϕ-Quantile KNN**] Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a d-dimensional space, and a multi-valued query object Q, the ϕ-quantile KNN problem is to retrieve the set Φ_K of K objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $d_\phi(Q, U) \leq d_\phi(Q, U')$.

[**ϕ-Quantile Group-base KNN**] Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a d-dimensional space, and a multi-valued query object Q, the ϕ-quantile group-base KNN problem is to retrieve the set Φ_K of K objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $gbd_\phi(Q, U) \leq gbd_\phi(Q, U')$.

Problem V: Indexing Structure for Uncertain Space

The aim is to build an efficient index to support various queries which rely on efficient processing or range query, such as size estimation of range query and similarity join. The index supports uncertain objects with arbitrary PDFs and is not sensitive to the size and shape of the query regions.

1.4 Contributions

The contributions of this thesis can be summarized as follows.

Contributions on the Probabilistic Top-k Skyline Operator

- We present a concrete model for the problem for both discrete and continuous cases.
- We develop an efficient, threshold-based algorithm to compute the exact top-k skyline objects. The algorithm is based on a set of novel techniques to calculate skyline probabilities and prune objects.
- To address the applications with a large number of instances per object or a given continuous probability density function (PDF) per object, we develop an efficient randomized algorithm with an accuracy guarantee, ϵ-approximation. It follows the framework of our exact algorithm to effectively remove non-top-k skyline objects.

Contributions on Probabilistic Skyline Operator over Sliding Windows

- We characterize the minimum information needed in continuously computing probabilistic skyline against a sliding window.
- We show that the volume of such minimum information is expected to be bounded by logarithmic size in a lower dimensional space regarding a given window size.
- We develop novel, incremental techniques to continuously compute probabilistic skyline over sliding windows.
- We extend our techniques to support multiple pre-given probability thresholds, as well as "top-k" probabilistic skyline.

Contributions on Probabilistic Top-k Dominating Queries

- We formally define a top-k dominating query on uncertain data with a given probability threshold imposed to support different confidence requirements.
- An efficient, threshold-based exact algorithm is proposed to take an advantage of the threshold-based paradigm [FLN03]. Based on a novel application of laws of large numbers [Gol01] and mathematic characterizations, a set of novel, effective pruning techniques have been proposed to pursue efficiency.
- We develop an efficient randomized algorithm with an accuracy guarantee. Novel processing techniques and data structures are developed in our randomized techniques.

Contributions on Quantile-based KNN over Multi-valued Objects

- We make the first attempt to identify KNN sensitive to the relative distributions among multi-valued objects.
- Efficient, novel techniques are proposed for computing quantile distance based KNN against a set of multi-valued objects and a given query object that is also multi-valued.
- We show that the problem of KNN against the quantile group-base distance is NP-hard. Novel and efficient algorithms are proposed with the approximation ratio 2.

Contributions on Indexing Structure for Uncertain Space

- A space-efficient index structure for organising multidimensional uncertain objects, UI-tree, is proposed. UI-tree can support arbitrary PDF of uncertain objects.

- We develop efficient solutions for various types of queries based on UI-tree, including range query, size estimation of range query, probabilistic top-k range query and similarity join.
- We provide rigorous analysis to estimate the filtering capacity of UI-tree.
- Extensive experiments over real and synthetic data sets are conducted to demonstrate the efficiency and scalability of UI-tree compared with other state-of-the-art techniques.

1.5 Thesis Organization

A thorough survey of existing techniques for managing uncertain data is presented in Chapter 2. We introduce current modeling and querying techniques, as well as database management systems designed especially for uncertain data.

In Chapter 3, we study the problem of top-k skyline on uncertain data. The problem is firstly formally defined for both continuous and discrete cases. Exact and random algorithms are proposed thereafter. Experiments on both real and synthetic data show the efficiency and effectiveness of the technique.

We extend our study of skyline operator on uncertain data to sliding windows in Chapter 4. After formally defining the problem, we characterize the candidate set with minimum size and give a formal proof of correctness. Efficient techniques based on R-tree structures are proposed. Extensive experiments on both real and synthetic datasets demonstrate that techniques proposed are scalable and support high speed data streams.

Besides skylines, dominating query is another important tool for multi-criteria decision making. In Chapter 5, we study the important problem of top-k dominating query on uncertain data. After giving a formal definition of the problem, both exact and random algorithms are presented. These are followed by experimental study to demonstrate the efficiency and effectiveness of our study on both real and synthetic datasets.

Another advanced query type, KNN query, is tackled in Chapter 6. Having observed that existing techniques for KNN processing on uncertain data may lose the important information of instances distribution, we define KNN queries sensitive to the instance distribution among multi-valued objects based on quantile distances. Techniques for quantile based KNN queries are then presented. As the problem of KNN against the quantile group-base distance is NP hard, in the following, we develop approximate algorithms with accuracy guarantee. Efficiency, accuracy, scalably and power of pruning rules are studied in experimental study.

Observing some deficiencies of existing indexing techniques for uncertain data, we investigate the problem of indexing multidimensional uncertain data in Chapter 7. The structure of the index UI-index is firstly introduced. Then, we show how to support various query types using it. Comprehensive experimental studies are conducted to demonstrate the efficiency and scalability of UI-index.

The thesis concludes in Chapter 8. Future work directions are also presented.

Chapter 2

Related Work

In this chapter, we overview the related work on uncertain data management. Firstly, we give a brief introduction of models for uncertain data. Next, we summarize existing techniques on various probabilistic query types. Then, we describe DBMSs specially designed for supporting uncertainty management. In the end, other research topics on uncertain data management are discussed, such as privacy, uncertain XML, clustering and mining, etc.

2.1 Modeling Uncertainty

The uncertainty of an object can be specified by three models [TXC07]: fuzzy model [GUP05], evidence-oriented model [Lee92, LSS96] and probabilistic model [SBHW06]. In fuzzy models, fuzzy entities, fuzzy attributes, fuzzy relationship, fuzzy aggregation, fuzzy constraints, etc are used to model uncertainty and imprecision. In evidence-oriented models, the Dempster-Shafer Theory of Evidence is applied to model uncertainty and imprecision. Probabilistic models specify uncertainty with probability values. In this chapter we focus on probabilistic models since it is is not only the most widely used but also it is the only model adopted in existing DBMSs for uncertainty analysis.

2.2 Possible World Semantics and Probabilistic Models

2.2.1 Possible World Semantics

We first introduce possible world semantics for object-based uncertainty. Suppose in a set of uncertain objects $\mathcal{D}$, an object U has probability $P(U)$ ($P(U) > 0$) to occur and all objects are independent. A *possible world* W is a subset of $\mathcal{D}$. Clearly, the occurrence probability of a possible world is $P(W) = \prod_{U \in W} P(U) \cdot \prod_{U \notin W}(1 - P(U))$. Note that an object with occurrence 1 must appear in any possible world. Let $\mathcal{W}$ be the set of all possible worlds of $\mathcal{D}$ and N be the number of objects with occurrence probability smaller than 1, then $|\mathcal{W}| = 2^N$. The sum of the membership probabilities of all possible worlds in $\mathcal{W}$ equals to 1; that is, $\sum_{W \in \mathcal{W}} P(W) = 1$.

For instance-based uncertainty, given a set of uncertain objects $\mathcal{U} = \{U_1, \cdots, U_n\}$, each uncertain object consists of a set of instances. All objects are independent. A *possible world* $W = \{u_1, \cdots, u_n\}$ is a set of instances with one instance from each uncertain object. The probability of W to appear is $P(W) = \prod_{i=1}^{n} p_{u_i}$. Let $\mathcal{W}$ be the set of all possible worlds, then $\sum_{W \in \mathcal{W}} P(W) = 1$.

General Model. In a general case, records in a data set may be correlated. A comprehensive study of possible world semantics is conducted by Hua *et al* in [HPZL08a, SIC07, YLKS08]. A set of records R_1, ..., R_m are *exclusive* if at most one of them could appear in a possible world and $\sum_{1 \le i \le m} P(R_i) \le 1$ where $P(R_i)$ is the occurrence probability of R_i. A set of exclusive records are also called a *generation rule* $\mathcal{R}$. Occurrence probability of a generation rule $\mathcal{R}$ is the sum of probabilities of all the records involved in $\mathcal{R}$; that is, $P(\mathcal{R}) = \sum_{R \in \mathcal{R}} P(R)$. Note that a generation rule (virtually regarded as an object) could contain only one record and different generation rules are independent. Given a set of m generation rules – $\mathcal{G}_D = \{\mathcal{R}_1, ..., \mathcal{R}_m\}$, a possible world W is defined as an element in $\prod_{\mathcal{R} \in \mathcal{G}'} \mathcal{R}$ where $\mathcal{G}'$ is a subset of $\mathcal{G}_D$, $\mathcal{G}'$ contains every generation rule $\mathcal{R}$ such that $P(\mathcal{R}) = 1$. Let $|\mathcal{R}|$ be the number of records in $\mathcal{R}$. The number of all possible worlds with respect to $\mathcal{G}_D$ is:

$$|\mathcal{W}| = \prod_{\mathcal{R} \in \mathcal{G}_D, P(\mathcal{R})=1} |\mathcal{R}| \prod_{\mathcal{R} \in \mathcal{G}_D, P(\mathcal{R})<1} (|\mathcal{R}| + 1) \tag{2.1}$$

Occurrence probability of a possible world W is:

$$\begin{aligned} P(W) &= \prod_{\mathcal{R} \in \mathcal{G}_D, \mathcal{R} \cap W \neq \phi} P(\mathcal{R} \cap W) \times \\ & \prod_{\mathcal{R} \in \mathcal{G}_D, \mathcal{R} \cap W = \phi} (1 - P(\mathcal{R})) \end{aligned} \tag{2.2}$$

where $P(\mathcal{R} \cap W)$ refers to the occurrence probability of a record which is in both $\mathcal{R}$ and W.

As an example, Table 2.1 records the ID of speeding vehicles (*Vehicle* in the table) and speed (*Speed*) captured by sensor nodes (*SID*) at certain location (*Loc.*) and time (*Time*). Each record is given an occurrence probability (P) representing its confidence to be true. In this example, records R1 and R2 can not appear in the same possible world; that is, R1 and R2 are exclusive. R3 is independent with them; this means the generation rule containing R3 only is independent with generation rule $\{R1, R2\}$. There are 6 possible worlds in all for this uncertain database, as shown in Table 2.2, along with corresponding occurrence probabilities.

RID	**SID**	**Time**	**Loc.**	**Vehicle**	**Speed**	**P**
$R1$	$S1$	$2:00PM$	$L1$	$HB1235$	120	0.7
$R2$	$S1$	$2:00PM$	$L1$	$HB1238$	150	0.2
$R3$	$S6$	$3:45PM$	$L17$	$HA2568$	170	0.9

Table 2.1: Speeding Vehicles Records.

Possible World	**Occurrence Probability**
$W_1 = \{\phi\}$	0.01
$W_2 = \{R1\}$	0.07
$W_3 = \{R2\}$	0.02
$W_4 = \{R3\}$	0.09
$W_5 = \{R1, R3\}$	0.63
$W_6 = \{R2, R3\}$	0.18

Table 2.2: Possible Worlds of Table 2.1.

2.2.2 Other Advanced Models

There are a number of advanced models. In this subsection, we introduce three representatives.

- Fuhr and Rolleke model uncertainty based on non-first-normal-form (NF2) [FR07] where records in a relation are assigned probabilistic weights. Imprecise attribute values are modelled as a probabilistic sub-relation. Moreover, a probabilistic relational algebra (PRA) is proposed as a generalization of standard relational algebra.

  Tuple t in a probabilistic relation modeled by NF2 contains three aspects, its attribute values, an event expression $t.\eta$ and event probability $t.\beta$. As shown in Figure 2.1 [FR07], the relation *BOOK* consists of atomic attributes *BNO*, *YEAR*, and attributes *PRICE*, *INDEX*, *AUTHOR* which are modeled by subrelations. Types of probabilistic relations include "deterministic", "independent", "disjoint" and "dependent". For example, subrelation *price* is

BOOK												
η	β	BNO	YEAR	PRICE			INDEX			AUTHOR		
				η	β	VAL	η	β	TERM	η	β	NAME
BÊ	1.0	1	92	BÊP1	0.6	30	BÊI1	0.9	IR	BÊAÊ	1.0	Smith
				BÊP2	0.4	25	BÊI2	0.8	DB	BÊAÊ	1.0	Jones
BÊ	1.0	2	93	BÊP3	1.0	29	BÊI3	0.9	AI	BÊAÊ	1.0	Miller
							BÊI4	0.8	DB			
BÊ	1.0	3	92	BÊP4	0.7	28	BÊI5	0.9	DB	BÊAÊ	1.0	Jones
				BÊP5	0.3	25						
BÊ	1.0	4	90	BÊP6	0.5	32	BÊI6	0.9	DB	BÊAÊ	1.0	Jones
				BÊP7	0.5	28						

Figure 2.1: Relation BOOK

disjoint meaning that one and only one event between $B\hat{E}P1$ and $B\hat{E}P2$ can be true. *INDEX* is *independent* meaning that both $B\hat{E}I1$ and $B\hat{E}I2$ can be true with different confidence and $AUTHOR$ is *deterministic* indicating that both values for *NAME* in this subrelation takes the same event probability as the tuple it belongs to, namely, 1.0.

Clearly, general model introduced earlier can also be used to model such NF2 probabilistic relations, in a clearer and more concise way.

- Sarma *et al* integrate *lineage* to model uncertainty [SBHW06]. Linage is associated with a data item carrying information about its derivation. A model ULDBs (Uncertainty-Lineage Databases) is developed by extending standard SQL relational model with the following four aspects [BSHW06a].

  1. *alternatives* capturing the uncertainty of contents of a tuple.
  2. *maybe* annotations "?" representing the uncertainty about the presence of a tuple.
  3. *confidence* values quantifying the degree of above two types of uncertainties.
  4. *lineage* recording derivation information of tuple alternatives.

In fact, besides a new ingredient *lineage*, this model is almost identical to the independent model in object-level uncertainty.

Table 2.3 gives an example of ULDBs. Record R1 can be either of the two tuples with different confidence values. R2 exists in this table with confidence 0.9. Table 2.4 captures vehicle ID and driver names. We join these two tables and project on the driver attribute, clearly obtaining only one tuple (John). We call this tuple R5. *Lineage* captures how R5 is derived from the original two tables by a function λ over the alternatives of tuples. $\lambda(R5, 1) = ((R1, 2), (R3, 1))$ means that the first alternative of R5 is derived from joining of the second alternative of R1 and first alternative of R3.

RID	**(Vehicle ID, Speed)**
R1	(HB1235, 120): 0.7 || (HB1238, 120): 0.2
R2	(HA2568, 170): 0.9 ?

Table 2.3: Vehicle and Speed.

RID	**(Vehicle ID, Driver)**
R3	(HB1238, John)
R4	(HC2457, Wendy)

Table 2.4: Vehicle and Driver.

- Sen and Deshpande utilize a probabilistic graphical model [Pea88] to facilitate query evaluation over uncertain data with general forms of correlations [SD07]. Besides *independence* and *mutual exclusivity*, *implies* and *nxor* are also explored; that is, the presence of one tuple implies absence of other tuples and high positive correlation between two tuples, respectively. Each tuple is associated with a boolean valued random variable X_t, namely *false* and *true*. In the probabilistic graphical model, nodes represent random variables while edges represent correlations. Thus different types of correlations, such as complete independence, mutual exclusivity, positive correlation can be modelled. Query evaluation problem with correlations is then transformed into equivalent problem under probabilistic graphical model and can be solved using existing techniques such as inference algorithms.

A lot of research work aims to represent uncertainty besides what we introduced above, for instance [AKG87, AKO07, BGMP92, DS96, Fuh90, IJ84].

2.3 Probabilistic Queries

We review existing work on important probabilistic queries in this subsection.

2.3.1 Query Evaluation

Cheng *et al* present a broad classification of probabilistic queries over one-dimensional uncertain data as well as techniques for evaluating probabilistic queries [CKP03]. There are four types in all, value-based non-aggregates, entity-based non-aggregates, value-based aggregates and entity-based aggregates according to two aspects: 1) the query requires qualifying objects or values and 2) the query is aggregate-based or not. An example of entity-based non-aggregate query is: given an interval $[l, r]$ where $l < r$, return a set of tuples (T_i, P_i) where attribute a of T_i is within the range $[l, r]$ with non-zero probability P_i. Bounding and pruning techniques are deployed to evaluate these queries. In [CXP$^+$04], Cheng *et al* explore access methods to also support range search for one dimensional data only.

A series of work has been done by Dalvi and Suciu from University of Washington to evaluate probabilistic queries. In [DS04], they tackle the problem of evaluating queries with uncertain predicates. Optimization algorithms that can evaluate efficiently most queries are presented. They also show that the evaluation of some queries is *#P-complete*; these queries are approached in two different methods: a heuristic avoiding significant errors and a Monte-Carlo simulation algorithm with precision guarantees. In [DS05], they propose to answer queries from statistic and probabilistic views. In [DS07a] a very clean and complete theoretical result is provided that the complexity of evaluating conjunctive queries over uncertain data set is either $PTIME$ or *#P-complete*.

Sen and Deshpande utilize probabilistic graphical model to approach the same problem in uncertain data sets with correlated tuples as introduced in Section 2.1 [SD07].

2.3.2 Aggregate Queries

A most recent work on aggregate query processing over relational uncertain data is from Stanford InfoLab as a function supported by their system Trio [MW07]. Five types of aggregate operators are tackled, *COUNT*, *MIN*, *MAX*, *SUM* and *AVG*. Among them, *COUNT*, *MIN* and *MAX* are relatively easy and there exist polynomial algorithms [HPZL08a]. However, it is shown in [DS07b] that results for *SUM* and *AVG* may be different in each possible world and computing *SUM* or *AVG* is #P-complete. Three approximate alternatives are proposed to avoid exhaustively materialize all possible results caused by "exact" aggregation in uncertain databases: lowest possible value, highest possible value and expected value. For instance, lowest possible value of SUM (*LSUM*) is defined as the sum of lowest value from each uncertain object (e.g., sets of tuples from probabilistic table that are governed by a generation rule). Specifically, expected-average (*EAVG*) value is approximated using expected-sum (*ESUM*) divided by expected-count (*ECOUNT*). Transformed aggregate queries are processed using TriQL techniques used in Trio which is an extension of SQL.

A thorough and fundamental study of OLAP against uncertain and imprecise data has been conducted in [BDRV05]. Other major work may be found in [CCT96, JKV07, MSS01, RSG05, RB92, SM03].

2.3.3 Join Queries

Join queries over one dimensional uncertain data are defined by Cheng *et al* in [CSP06] in a continuous case. Uncertainty over a data item a is parameterized with an uncertainty interval $a.U$ and PDF $a.f(x)$. Uncertainty comparison operators, *equality*, *inequality*, *greater than* and *less than* are defined in a continuous fashion. Take *equality* between two uncertain items a and b as an example. Since the PDFs for both a and b are continuous, the probability that a equals b could be infinitesimally small. A new parameter *resolution* (c) is introduced to avoid this: a equals b if they are within c distance i.e., $|a - b| \leq c$. The probability that a equals b with resolution c is defined as:

$$P(a =_c b) = \int_{-\infty}^{+\infty} a.f(x) \cdot (b.F(x + c) - b.F(x - c))dx \tag{2.3}$$

where $b.F(x)$ denotes the cumulative distribution function (CDF) of b.

Denote θ_u as a uncertainty comparison operator and R and S are uncertain data sets; probabilistic join query (PJQ) returns all pairs of tuples (R_i, S_j) with $P(R_i \theta S_j) > 0$, where $R_i \in R, S_j \in S$. Probabilistic threshold join query (PTJQ) further imposes a probability threshold and only uncertain item pairs with matching probability value no less than this threshold satisfy PTJQ. Based on this threshold, pruning techniques in different indexing levels are proposed to answer PTJQ.

A recent work on join queries on uncertain data is given in in a top-k fashion [AW07] by Agrawal and Widom. In such confidence-aware joins, only results with top-k matching confidence will be output.

2.3.4 Similarity Joins

Kriegel *et al* study similarity joins on uncertain spatial objects in [KKPR06]. The probability that distance between two uncertain objects U and V is within a range $[a, b]$ is defined as,

$$P(a \leq d(U,V) \leq b) = \int_a^b f_d(U,V)(x)dx \tag{2.4}$$

where $f_d(U,V)$ is the probabilistic distance function between U and V. Although $f_d(U,V)$ may be computed directly for some uncertain object representations, for efficiency reasons, Kriegel *et al* propose algorithms based on Monte-Carlo sampling technique where each uncertain object is represented by a set of s sampled instances. In this case, the similarity join probability between U and V is defined as follows. Assume all instances in an uncertain object take the same probability to appear

$$P(d(U,V) \leq \epsilon) = \frac{|\{(u_i, v_j)|d(u_i, v_j) \leq \epsilon, 1 \leq i, j \leq s\}|}{s^2}$$

Here $P_{d(u_i,u_j)\leq\epsilon}$ denotes the probability that the distance between u_i and v_j is not greater than ϵ, where $u_i \in U$ and $v_j \in V$. As shown in Figure 2.2 where there are totally 9 pairs of instances, only the distances of three pairs of instances connected with solid line are smaller than a given distance threshold; consequently $P(d(U,V) \leq \epsilon) = 1/3$. In their algorithms, instances are also grouped and indexed using R-tree. Then effective pruning techniques based on ϵ are applied. For any two input uncertain objects and distance threshold ϵ, the similarity join probability between these two objects regarding ϵ will be output by their algorithms.

In [BPS06a], a similar problem $-$ *similarity matching* is investigated. In their settings, uncertainty of feature vectors follow Gaussian distribution. A novel index structure, Gauss-tree, is developed for similarity matching processing.

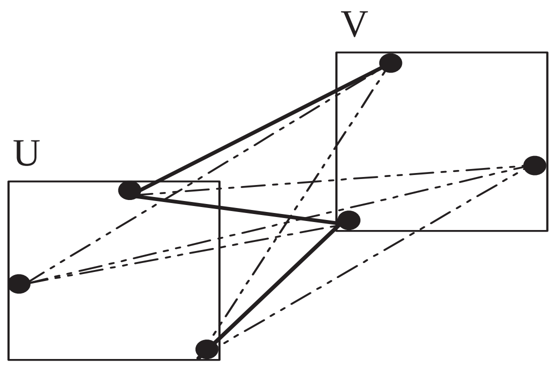


Figure 2.2: Probabilistic Similarity Join

2.3.5 Top-k Queries

Top-k queries are important in analyzing uncertain data. Unlike a top-k query over certain data which returns the k best alternatives according to a ranking function, a top-k query against uncertain data has inherently more sophisticated semantics. Soliman *et al* [SIC07] first relate top-k queries with uncertain data. They define two types of important queries - *U-Topk* and *U-kRank*, regarding discrete cases.

U-Top$k$ returns a set of k records which as a whole have the highest probability to be the top-k results in all possible worlds. A precise definition is as follows [SIC07].

> Let D be an uncertain data set with possible worlds space $\mathcal{W} = \{W_1, ..., W_n\}$. Let $\mathcal{T} = \{T^1, ..., T^m\}$ be a set of k-length record vectors, where for each $T^i \in \mathcal{T}$: (1) records of T^i are ordered according to scoring function $\mathcal{F}$, and (2) T^i is the top-k answer for a non empty set of possible worlds $W(T^i) \subseteq \mathcal{W}$. A *U-Topk* query, based on F, returns $T^* \in \mathcal{T}$, where $T^* = argmax_{T^i \in \mathcal{T}}(\sum_{w \in W(T^i)} P(w))$.

U-kRank retrieves k ordered records where the i-th record has the highest probability of ranking in the i-th position among all possible worlds [SIC07].

> Let D be an uncertain data set with possible worlds space $\mathcal{W} =$

$\{W_1, ..., W_n\}$. For $i = 1, ..., k$, let $\{x_i^1, ..., x_i^m\}$ be a set of records, where each record x_i^j appears at rank i in a non empty set of possible worlds $W(x_i^j) \in \mathcal{W}$ based on scoring function $\mathcal{F}$. A *U-kRanks* query, based on $\mathcal{F}$, returns $\{x_i^*; i = 1, ..., k\}$, where $x_i^* = argmax_{x_i^j}(\sum_{w \in W(x_i^j)} P(w))$.

Methods proposed in [SIC07] navigate all possible states of the search space, meanwhile minimizing the number of tuples accessed. Based on novel observations, Yi *et al* [YLKS08] significantly improve the efficiency while tackling the same queries.

Threshold based top-k queries defined by Hua *et al* [HPZL08a, HPZL08b] aim to retrieve all records whose probability of being top-k results in all possible worlds is no less than a given probability threshold. Re *et al* [CR07] deal with query evaluation on probabilistic database and results are ranked according to the probability of satisfying a given query.

Expected rank is proposed to use for answering top-k queries on uncertain data by Cormode *et al* in [CLY09]. By utilizing expected rank, some fundamental properties such as *exact-k*, *containment*, *unique rank*, *value invariance* and *stability* are all satisfied. These properties naturally hold on certain data but are missed by some top-k definitions on uncertain data. In [GZM09], Ge *et al* study the score distribution and typical answers of top-k queries on uncertain data. By presenting the score distribution of all top-k vectors, the users are able to choose among all results along the score-probability dimensions, as shown in Figure 2.3 [CLY09]. Instead of displaying distributions of all potential top-k vectors, the authors also propose to provide a a number of typical vectors that effectively sample this distribution.

Finally in [LSD09], Li *et al* propose a universal ranking function based on computing the parameters of generating functions. Although this universal ranking

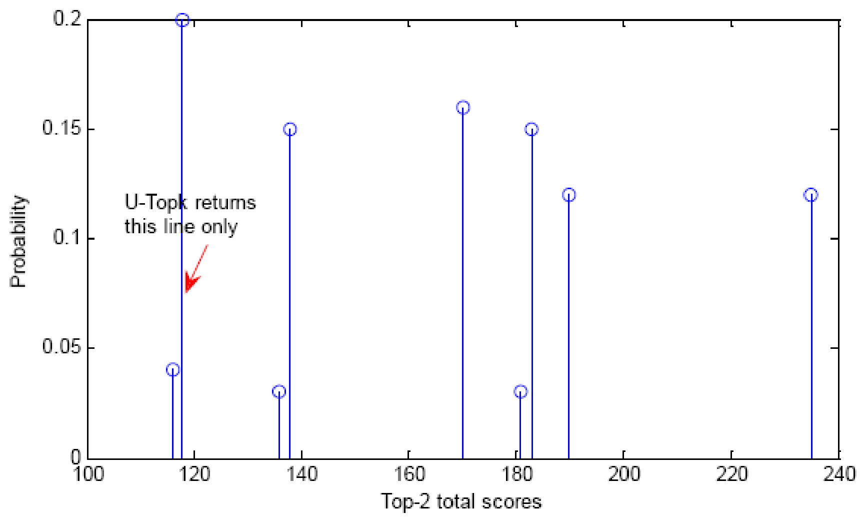


Figure 2.3: The Distribution of Top-k Tuples' Total Scores

function generalizes most existing probabilistic ranking semantics, it does not support certain definition with specific requirements like all top-k tuples should belong to the same possible world, e.g. *U-Topk*.

2.3.6 Nearest Neighbor Queries

The problem of nearest neighbor query on uncertain objects is tackled in [KKR07]. In a discrete case, both uncertain objects and a query object are represented by a set of s sampled instances. The probability that uncertain object U is the nearest neighbor of query object Q $P_{nn}(U, Q)$ is defined based on the instance pairs from U and Q.

$$P_{nn}(U, Q) = \frac{\sum_{i,j \in 1 \ldots s} P_{nn}(u_i, q_j)}{s^2}$$

where $P_{nn}(u_i, q_j)$ is the probability that instance $u_i \in U$ is the nearest neighbor of instance $q_j \in Q$. $P_{nn}(U, Q)$ in continuous cases is computed based on the probabilistic distance between Q and U and the probabilistic distance between Q and other objects except U.

To facilitate query processing, instances inside an object are clustered into several groups bounded by minimal bounding boxes (MBRs) and indexed by R-tree. Thus higher level pruning and validating measures can be applied.

Constrained nearest neighbor query is studied in [CCMC08] with a pre-given probability threshold. Only objects with probability no less than this threshold of being nearest neighbor of the query object will be output. In [BSI08], Beskales *et al* define the top-k probabilistic KNN problem based on possible world semantics and develop techniques optimizing both I/O and CPU cost.

2.3.7 Skyline Queries

For two points u and v in a multi-dimensional space, u *dominates* v ($u \prec v$) if in each dimension the coordinate of u is not greater than that of v and there is one dimension in which the coordinate of u is smaller than v. For a given data set, the *skyline* operator returns all points in the data set which are not dominated by other points. As illustrated in Figure 2.4, skyline points are a, b and c since they are not dominated by any other points. Skyline operator over uncertain objects is more complex since it involves sophisticated analysis of probability distribution of each uncertain object. As in Figure 2.5, generally instances in each uncertain object have different dominating ability. This problem is firstly approached by Pei *et al* in [PJLY07]. In a continuous case, suppose that f is the PDF of uncertain object U in the data space $\mathcal{D}$, the probability for U to be a skyline object is:

$$Pr(U) = \int_{u \in \mathcal{D}} f(u) \prod_{\forall V \neq U} (1 - \int_{v \prec u} f'(v) dv) du \tag{2.5}$$

Here $\prod_{V \neq U}(1 - \int_{v \prec u} f'(v)dv)$ is the probability that the point $u \in U$ is not dominated by any uncertain objects. f' denotes the PDF of V. In a discrete case, the skyline probability of U is:

$$Pr(U) = \sum_{u \in U} (P(u) \times \prod_{\forall V \neq U} (1 - \sum_{v \in V, v \prec u} P(v))) \tag{2.6}$$

$\prod_{V \neq U}(1 - \sum_{v \in V, v \prec u} p(v))$ is the probability that $u \in U$ is not dominated by any other objects. Recall that $P(u)$ denotes the appearance probability of instance u.

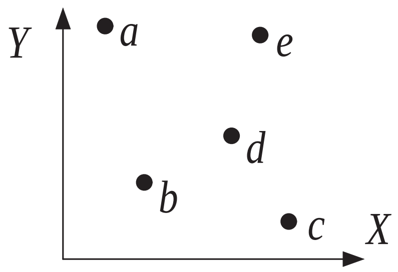


Figure 2.4: Certain Objects.

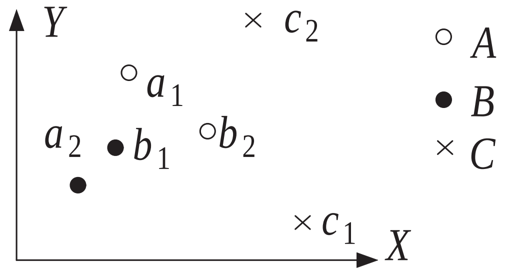


Figure 2.5: Uncertain Objects.

Bounding-pruning-refining iteration is deployed to achieve efficiency. Two algorithms, bottom-up and top-down, are developed. The bottom-up algorithm computes $P(U)$ from instance level. After calculating skyline probabilities of some selected instances, these values are used to prune other instances and objects. Top-down algorithm, on the other hand, partitions instances of one uncertain object into several groups and apply pruning techniques in the group and object level.

A variation of uncertain skyline, monochromatic and bichromatic reverse skyline search over uncertain objects, is studied in [LC08].

2.4 Indexing Uncertain Data

The first index structure supporting range queries on multi-dimensional spatial uncertain data with arbitrary PDFs is U-tree [TCX$^+$05]. U-tree is a novel modification of R-tree to facilitate a set of new pruning and validating techniques. A

d-dimensional uncertain object U is modeled using a d-dimensional uncertain region $U.ur$ and probability density function $U.pdf(x)$. Suppose the query region of a range query Q is r_Q, the appearance probability of U in r_Q is defined as:

$$P_{app}(U,Q) = \int_{U.ur \cap r_Q} U.pdf(x)dx \tag{2.7}$$

where $U.ur \cap r_Q$ is the intersection of $U.ur$ and r_Q. Given a probability threshold p, uncertain objects with $P_{app}(U,Q) \geq p$ are retrieved by the range query.

The basic idea to build a U-tree is illustrated in Figure 2.6 where polygon $U.ur$ is the uncertain range of 2-dimensional uncertain object U. For a given probability p_1, in each dimension, two lines are calculated. In the horizontal dimension, U has the probability p_1 to occur on the left side of line l_{1-}, also the probability p_1 to occur on the right side of line l_{1+}. Similarly, l_{2-} and l_{2+} are calculated in the vertical dimension. The shadowed region forms the probability constrained region (PCR) of U with respect to p_1. Such a region is used to prune or validate objects. There are multiple PCRs computed beforehand to facilitate range query processing, as shown in Figure 2.7. To tradeoff between space costs and pruning/validating abilities, a U-tree structure is constructed based on the approximation of such polygons.

In [BGK+07, BPS06b], Böhm *et al* study range queries with the constraint that instances of uncertain objects follow Gaussian distribution. Results are ranked according to the probability of satisfying range queries. A more recent work addressing indexing high dimensional uncertain data is [AY08].

Motivated by some shortcomings of U-tree, in Section 7 we also develop another index structure UI-tree.

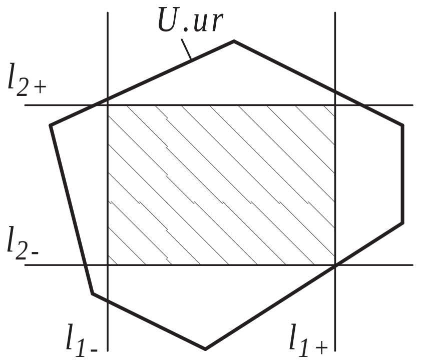


Figure 2.6: Pruning/Validating in U-tree.

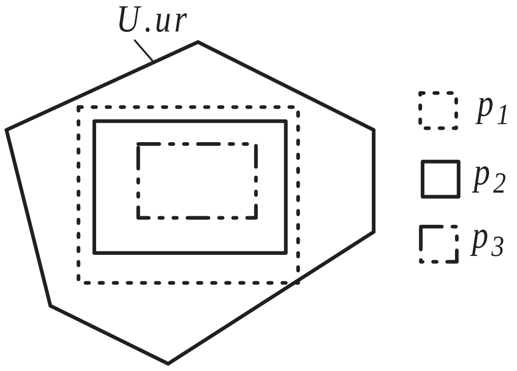


Figure 2.7: Multiple PCRs.

2.5 Uncertainty Management Systems

Many systems have been developed and implemented to support uncertain data management. We briefly summarize six representatives below.

2.5.1 Trio

Trio is developed by Standford InfoLab [ABS+06, BSHW06b, MTdK+07, Wid05]. As a database system, it not only tackles modelling and analyzing data but also the accuracy and lineage of data. Trio is developed based on data model ULDB which is introduced in Section 2.1. It is implemented on the top of traditional relational DBMS (*PostgreSQL*). Query language in Trio is an extension of SQL, TrioQL. TrioQL handles queries, as well as accuracy and lineage of data. Figure 2.8 illustrates the system architecture of Trio [MTdK+07]. The Trio API accepts TrioQL as well as regular SQL queries from client and translates them into standard SQL queries; in the relational DBMS, data tables are encoded, namely, are integrated with uncertainty information such as confidence and alternatives as introduced in Section 2.1. Trio Stored Procedures handles confidence and lineage information.

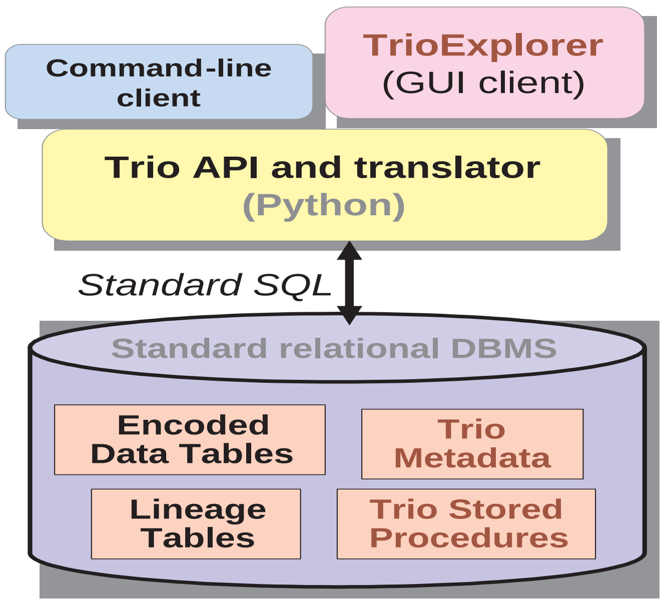


Figure 2.8: Trio System Architecture

2.5.2 MystiQ

MystiQ is a database system managing uncertainty in a probabilistic view developed by the University of Washington [BDM+05]. The system contains four main components, data modelling language (mDML), data definition language (mDDL), preprocessor and query translation engine. mDDL defines *approximate match operators* and allows users to specify *confidence* in query predicates. Below is an example query in mDDL [BDM+05].

SELECT F.title, D.name
FROM Director D, Films F
WHERE D.did = F.did
AND D.name ∼ 'Copolla' [CONFIDENCE = 0.9]
AND F.year ∼ 1975 [CONFIDENCE = 0.7]

This query retrieves film title and director name where the film was produced in approximately 1975 with confidence 0.7 and the director name is approximately 'Copolla' with confidence 0.9, from tables Director and Film. mDDL is integrated with the new components to manage uncertainty; the new components include predicate functions to specify measure used to generated similarity probabilities, global constraints for detecting and resolving inconsistency, etc. Based on mDDL specification, the preprocessor generates additional relational tables integrating probability values. The query translation engine is a critical component in MystiQ, translating queries written in mDML into regular SQL queries. Query evaluation techniques in MystiQ are introduced in [DS04, CR07].

2.5.3 URank

URank is a system mainly designed for answering *U-Topk* and *U-kRanks* queries from Waterloo University and UIUC [SFC07]. Querying techniques are introduced in [SIC07]. URank is also built on traditional relational DBMS with new components to cope with uncertainty. The system is composed of two layers as shown in Figure 2.9, *Storage Layer* and *Processing Layer*. Physical data and generation rules are manipulated in *Storage Layer*, as well as different access methods, such as random access and sorted access. *Processing Layer* is mainly based on techniques in [SIC07]. *Space Navigation* accesses data from the *Storage Layer*. Each accessed tuple is sent to the component *State Formulation* to calculate probabilities of newly generated states. *Rule Engine* is the part of the system to handle such state probability computation based on generation rules.

2.5.4 MayBMS

MayBMS project, developed by Cornel University, aims to build a probabilistic database management system based on the Postgres server. MayBMS is founded

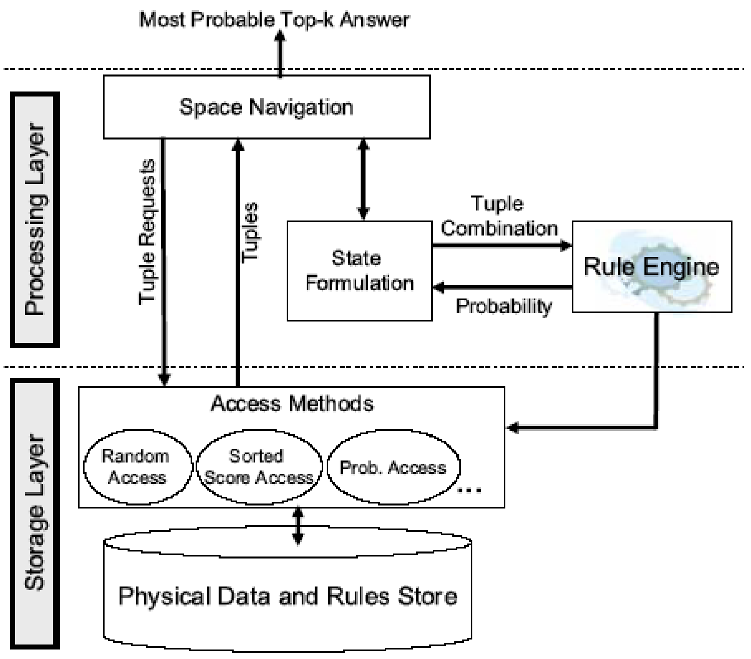


Figure 2.9: URank Framework

on a series of theories and principles to extend mature relational database technologies to create robust and scalable systems for managing and querying large scale uncertain datasets. Research themes of MayBMS include data representation, storage mechanism, query language, query processing techniques, query optimization, update language, concurrency control and recovery, APIs, etc. Figure 2.10 [AK08] illustrates the architecture of MayBMS system. Compared to traditional DBMSs, systems for managing uncertain data need to deal with two additional levels of abstraction, the representation system and the possible worlds model [AK08].

2.5.5 ORION

The ORION project is developed by researchers in Purdue University. Orion supports both attribute and tuple uncertainty with arbitrary correlations. Contributions of ORION include modeling, access methods, query optimization, graphical

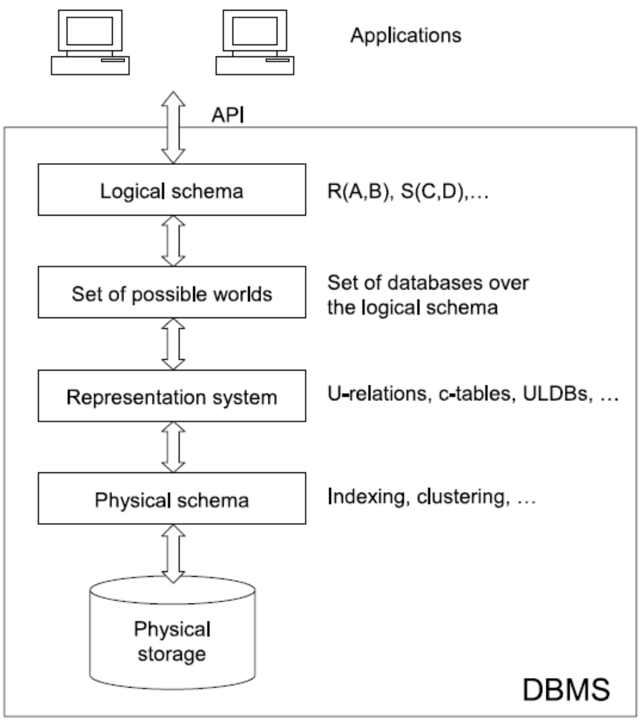


Figure 2.10: MayBMS Architecture

visualization of and statistical inference over uncertain data. The fundamental difference between Orion and related projects is indeed its support of attribute-level continuous uncertainty, which enables the system to represent probabilistic data in a natural and efficient manner. Figure 2.11 [SMM+08] shows the architecture of the entire system. In the figure, *uncertain type*, *system catalog* in system level, *uncertain data types*, *uncertainty query functions* in user level are new components that correspond to the primary features of the Orion data model. *Cost estimation functions*, *indexes* and *query engine* in system level are portions of the PostgreSQL backend that are extended to support queries over uncertain data. Other parts in the architecture (which include the majority of the PostgreSQL backend) indicate components that are not modified.

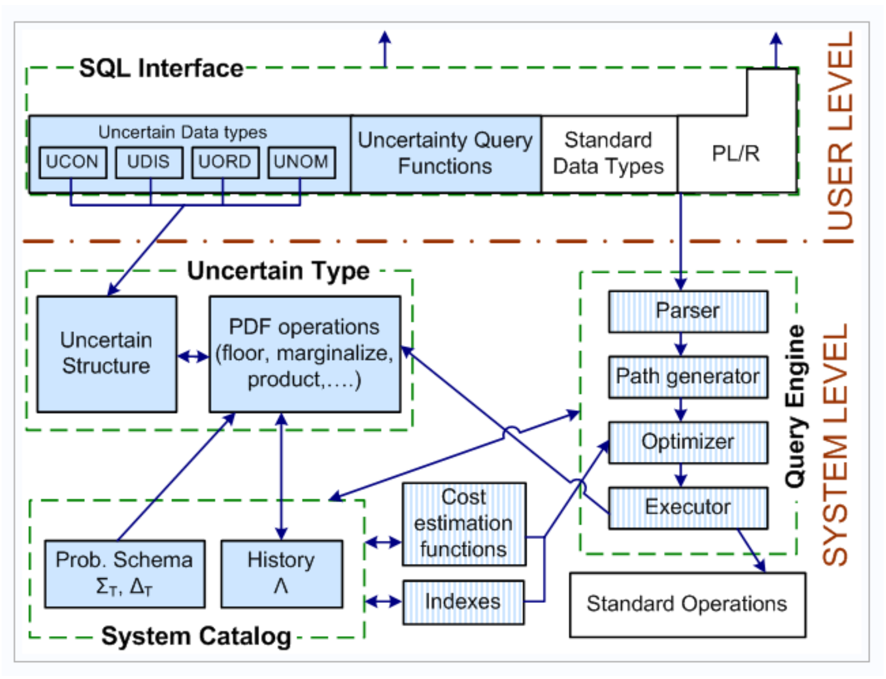


Figure 2.11: ORION 2.0 Architecture

2.5.6 MauveDB

MauveDB, developed by University of Maryland, aims to provide abstractions to continuously apply statistical modeling tools to streaming sensor data. MauveDB supports a new abstraction called ”model-based views” to achieve the above goal, where a model-based view is analogous to a traditional database view in that it can be used to present a consistent “view” of the underlying data to the user. The architecture of the system is shown in Figure 2.12 [DM06]. As shown, MauveDB consists of three main modules, *storage manager*, *view manager* and *query processor*. The storage manager is responsible for maintaining the raw sensor data on disk, as well as maintaining indexes on the tables. The view manager tracks the type and status of the views in the system and then provides corresponding information to the query processor. The query processor mainly answers users' queries.

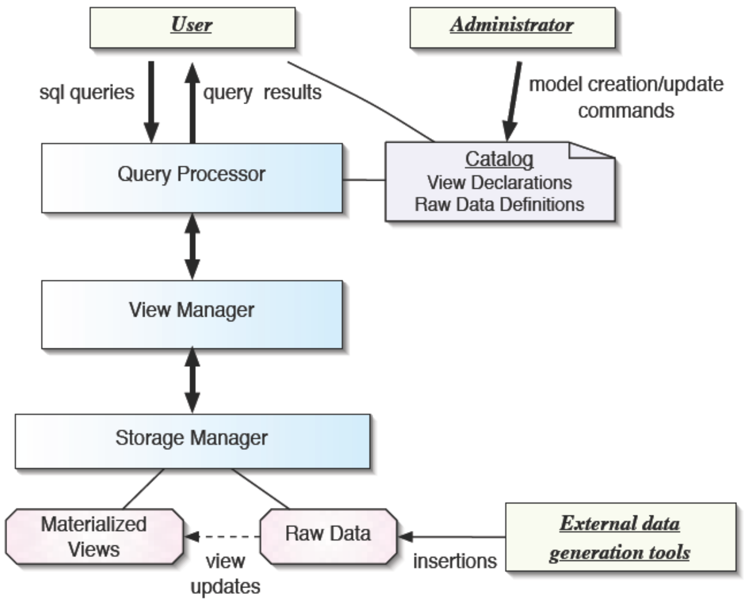


Figure 2.12: MauveDB Architecture

Other systems managing uncertain data may be found in [LLRS97, MSCP03, Mot88].

2.6 Other Related Work on Uncertainty Management

Clustering uncertain objects is addressed in [KP05]. The distance between two uncertain objects is the same as Equation(2.4). Key concepts in density-based clustering on uncertain objects, such as *core objects*, *core object probability*, and *reachability probability* among objects are defined where a core object has a dense neighborhood and both core object probability and reachability probability are derived based on the Equation(2.4), respectively. Novel density-based clustering algorithm $\mathcal{F}DBSCAN$ is then developed based on these new concepts.

Ngai *et al* address the same problem in [NKC+06] using clustering algorithm based on the traditional K-mean algorithm. Different from the probabilistic distance functions in [KP05], distance values used between a pair of uncertain objects or between an uncertain object and a cluster are *expected* values. For arbitrary PDF, such *expected values* often involve expensive numerical integration calculation. Pruning techniques are also proposed to avoid such an expensive step. Chau *et al* tackle the problem of mining uncertain data in [CCKN06] as an extension of the techniques proposed in [NKC$^+$06].

Query evaluation over uncertain data has also been studied against other applications, such as data streams [CG07], sensor networks [CP03, LCLC04], moving objects [CKP04], video retrieval [BGK$^+$07], XML data [AS06, HGS03b, HGS03a, KS07, NJ02, SA07], categorical data [SMP$^+$07], etc. Theoretical problems such as functional dependency [SUW08] and confidence computation [STW08] analysis over uncertain relation data have also been addressed. Privacy issues in uncertain semantics are studied in [Agg08, BAN08].

Chapter 3

The Probabilistic Top-k Skyline Operator [1]

Skyline analysis (e.g., [BKS01, CJT$^+$06, HJLO06, LYWL05, PJET05, SS06, TXP06, YLL$^+$05]) has been demonstrated very useful in multi-criteria decision making applications. In a multidimensional space where a preference order is given in each dimension (e.g., low price and high quality are preferred in dimensions price and quality), a point u_1 *dominates* another point u_2 if u_1 is not worse than u_2 in every dimension, and is better than u_2 in at least one dimension according to the preference. Point u is a *skyline* point if there is no any other point u' such that u' dominates u. Given a set of certain points, the skyline consisting of all skyline points presents all best possible tradeoffs among the dimensions under consideration.

Skyline analysis is also meaningful on uncertain data. To motivate our study, let us consider an example. Suppose that the service performance of a realtor regarding

[1]The techniques presented in this chapter originally appear in the paper "Ranking Uncertain Sky: the Probabilistic Top-k Skyline Operator", Ying Zhang, Wenjie Zhang, Xuemin Lin, Jian Pei, and Bin Jiang, *submitted to Journal of Information Systems (conditional acceptance)*

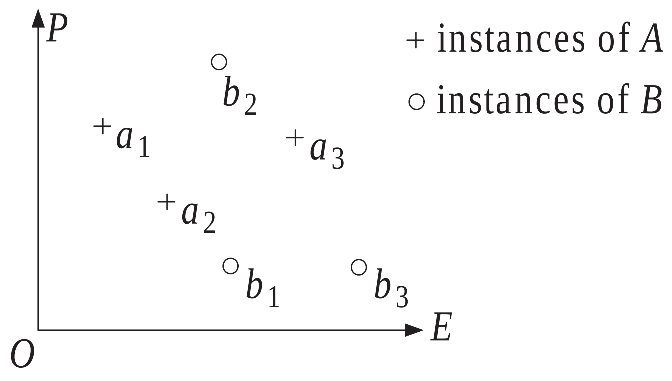


Figure 3.1: Instances of Two Realtor Uncertain Objects

a property sold by her/him is evaluated against the two aspects: percentage of the actual price increments against the reserve price (dimension P) - the higher the better, and the service quality ranked by the property owner (dimension E) where scores 1 to 5 are given for ranking purposes - the lower the better. Thus, the performance of a realtor can be naturally modeled as an uncertain object in the 2-dimensional space (P, E), and each successful sale record can be viewed as an instance of the uncertain object.

There can be a large number of realtors doing business in an area. Therefore, a new customer (property owner) often likes to receive recommendations of realtors who are good in both aspects - high value of P and lower value of E. Unfortunately, in practice it is often impossible for one realtor dominates all other realtors in both aspects against all sale records. Thus, skyline analysis against uncertain data makes sense here.

While skyline on certain data is well defined, finding skyline of realtors as uncertain objects is not straightforward. Consider two realtors A and B and their instances in Figure 3.1 where we record multiplicative inverse of the price incremental percentage on dimension P and assume that coordinate values are all positive without loss of generality. Instances a_1 and a_2 of A dominate instance b_2 of B.

Instance b_1 of B dominates instance a_3 of A. Moreover, a_1 neither dominates nor is dominated by b_1, b_3.

Clearly, A neither dominates nor is dominated by B completely. In fact, A takes a probability of $\frac{2}{9}$ dominating B, and B takes a probability of $\frac{1}{9}$ dominating A, assuming each instance takes the same probability to appear.

Generally, a realtor as an uncertain object takes a probability not being dominated by any other realtors. We are particularly interested in the *skyline probability* - the probability that a realtor is not dominated by any other realtors. The skyline probability can be used as a **quality measure** of the realtor's performance.

Note that computing the skyline probabilities of realtors is different from computing the skyline of realtors using aggregates (e.g., the average scores) on the instances (Table 2 in [PJLY07] illustrates such a difference). Using aggregates like the average scores, the distribution information of instances is lost. Contrarily, the skyline probability of a realtor considers the relative distribution of the instances of the realtor against instances of other realtors.

Due to a large number of realtors in the market, a new customer may often ask for top-k realtors who have the highest skyline probabilities. This is an example of computing top-k skyline objects from uncertain data. Clearly, the computation of top-k skyline objects is very useful in many other applications where ranking uncertain objects in a multi-dimensional space is involved; for instance, ranking NBA players based on their game-by-game statistics.

To the best of our knowledge, we are the first to identify the problem of computing top-k skyline objects on uncertain data. In this chapter, our investigation includes both discrete and continuous cases. Our principal contributions can be summarized as follows.

- We present a concrete model for the problem for both discrete and continuous cases.

- We develop an efficient, threshold-based algorithm to compute the exact top-k skyline objects. The algorithm is based on a set of novel techniques to calculate skyline probabilities and prune objects.

- To address the applications with a large number of instances per object or a given continuous probability density function (PDF) per object, we develop an efficient randomized algorithm with an accuracy guarantee, ϵ-approximation. It follows the framework of our exact algorithm to effectively remove non-top-k skyline objects.

We evaluate the effectiveness and the efficiency of our techniques for computing top-k skyline objects on uncertain data. Our experiment results demonstrate the efficiency of both algorithms and also show that the randomized algorithm is highly accurate in practice. Moreover, our algorithms can be immediately applied to finding uncertain objects with skyline probabilities above a given threshold (p-skyline). While the randomized algorithm is the first technique to compute p-skyline regarding the continuous case with an ϵ-approximation guarantee, our exact algorithm significantly outperforms the existing techniques in [PJLY07].

The remainder of this chapter is structured as follows. In Section 3.1, we model the problem and present preliminaries. Section 3.2 presents a framework to be adopted in the exact algorithm and randomized algorithm, as well as discrete and continuous cases. Section 3.3 applies the framework to our exact algorithm, while Section 3.4 presents novel techniques for the randomized algorithm. In Section 3.5, we extend our techniques to compute p-skyline. Section 3.6 reports the experiment results. We conclude this chapter in Section 3.7.

3.1 Background Information

3.1.1 Problem Statement

Points and/or instances referred in this chapter, by default, are in a d-dimensional numeric space $D = \{D_1, \cdots, D_d\}$ where D_i denotes the i-th dimension. For two points u and v, u *dominates* v, denoted by $u \prec v$, if $u.D_i \leq v.D_i$ for every $D_i \in D$, and there exists a dimension $D_j \in D$ where $u.D_j < v.D_j$. Given a set of points, the *skyline* consists of all points which are not dominated by any other point.

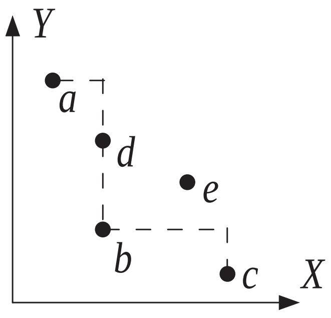


Figure 3.2: Certain Data.

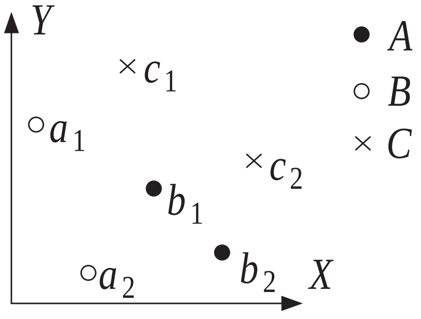


Figure 3.3: Uncertain Data.

Example 3.1. *Consider a set of points in Figure 3.2. The skyline consists of a, b, and c where b dominates d and e.*

3.1.2 Problem Definition

We investigate both discrete and continuous cases.

Discrete Case. Given a set of uncertain objects $\mathcal{U} = \{U_1, \cdots, U_n\}$, a *possible world* $W = \{u_1, \cdots, u_n\}$ is a set of instances with one instance from each uncertain object. The probability of W to appear is $Pr(W) = \prod_{i=1}^{n} p_{u_i}$. Let Ω be the set of all possible worlds, then $\sum_{W \in \Omega} Pr(W) = 1$.

We use $SKY(W)$ to denote the set of objects such that for each object $U \in SKY(W)$, U has an instance in the skyline of a possible world W. The probability that U appears in the skylines of the possible worlds is $P_{sky}(U) = \sum_{U \in SKY(W), W \in \Omega} P(W)$. $P_{sky}(U)$ is called the *skyline probability* of U.

Example 3.2. *Figure 3.3 plots a set of uncertain objects. Assume all instances take the probability* 0.5 *to appear. We have* 8 *possible worlds in total.* $P_{sky}(A) = 1$ *since* a_1 *and* a_2 *are in the skyline of every possible world.*

Note that c_1 *is dominated by every instance of* A *and* c_2 *is dominated by every instance of* B*; consequently, there is no possible world where* C *is in the skyline. Thus,* $P_{sky}(C) = 0$.

Note that B *is in the skylines of* 4 *possible worlds* $\{a_1, b_1, c_1\}$, $\{a_1, b_1, c_2\}$, $\{a_1, b_2, c_1\}$, *and* $\{a_1, b_2, c_2\}$. *Therefore,* $P_{sky}(B) = 4 \times (0.5 \times 0.5 \times 0.5) = 0.5$.

By the above definition, it can be immediately verified that for each instance u of $U \in \mathcal{U}$, the total probability of the possible worlds, in which instance u is in the skyline, is $p_u \times \prod_{V \in (\mathcal{U}-U)} \left(1 - \sum_{v \in V, v \prec u} p_v\right)$. Let

$$P_{sky}(u) = \prod_{V \in (\mathcal{U}-U)} \left(1 - \sum_{v \in V, v \prec u} p_v\right).$$

$P_{sky}(u)$ is called the *skyline probability* of u, which is a conditional probability computed when we only consider the possible worlds containing instance u. It is also immediate that the skyline probability $P_{sky}(U)$ of U can be re-written below,

$$\begin{aligned} P_{sky}(U) &= \sum_{u \in U} p_u \times P_{sky}(u) \qquad (3.1) \\ &= \sum_{u \in U} \Big(p_u \times \prod_{V \in (\mathcal{U}-U)} \left(1 - \sum_{v \in V, v \prec u} p_v\right)\Big). \end{aligned}$$

Continuous Case. Similarly, given a set of uncertain objects $\mathcal{U} = \{U_1, ..., U_n\}$ such that each U_i has a PDF f_{U_i} defined on U_i. The possible world semantic can be extended to cover the continuous case as follows. A *possible world* $W = \{u_1, u_2, ..., u_n\}$ is a point in the space $\Omega = \prod_{i=1}^{n} U_i$ such that $\int_{W \in \Omega} \prod_{i=1}^{n} f_{U_i}(u_i) du_1 du_2 ... du_n = 1$.

Similarly, we define $SKY(W)$ by including the objects with a point in the skyline of W. The skyline probability of U is

$$P_{sky}(U) = \int_{U \in SKY(W), W \in \Omega} \prod_{i=1}^{n} f_{U_i}(u_i) du_1 du_2 ... du_n. \tag{3.2}$$

This can be rewritten as:

$$P_{sky}(U) = \int_{u \in U} f_U(u) \prod_{V \neq U} (1 - \int_{v \prec u, v \in V} f_V(v) dv) du. \tag{3.3}$$

Problem Statement. In this chapter, we investigate the problem of finding top-k skyline objects on uncertain data (Top-k SOUND); it is formally defined below.

Definition 3.1 (Top-k SOUND)**.** *Given a set $\mathcal{U}$ of uncertain objects and an integer k, retrieve the k uncertain objects with the maximum skyline probabilities.*

Table 7.1 summarizes the notations used in this chapter.

Notation	Definition
U, V	uncertain objects
u, v	instances of uncertain objects
$f_U(u)$	probability density function of U
p_u	the probability of u to appear
$M(U)$	the weighted centroid of U, i.e., $\sum_{u \in U} p_u \times u$
$U.MBB$	the minimum bounding box of U
$U_{max}(U_{min})$	the upper (lower) corner of $U.MBB$
$P_{sky}(U)$	skyline probability of U
$P_{sky}(u)$	skyline probability of u
$P_{sky}(U)\|_{\mathcal{U}}$	skyline probability of U in $U \cup \mathcal{U}$
$Pr(U)$	the probability of U
$PD(U)$	the set of objects partially dominating U
n_U	the number of instances in U

Table 3.1: The Summary of Notations.

3.1.3 Preliminaries

Dominance Relationships. A pair U, V of uncertain objects may have three kinds of relationships as illustrated in Figure 3.4.

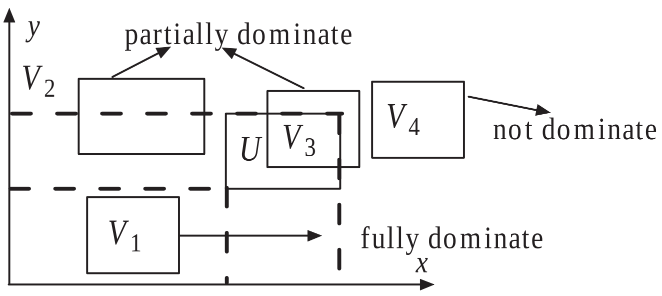


Figure 3.4: Dominance Relationships.

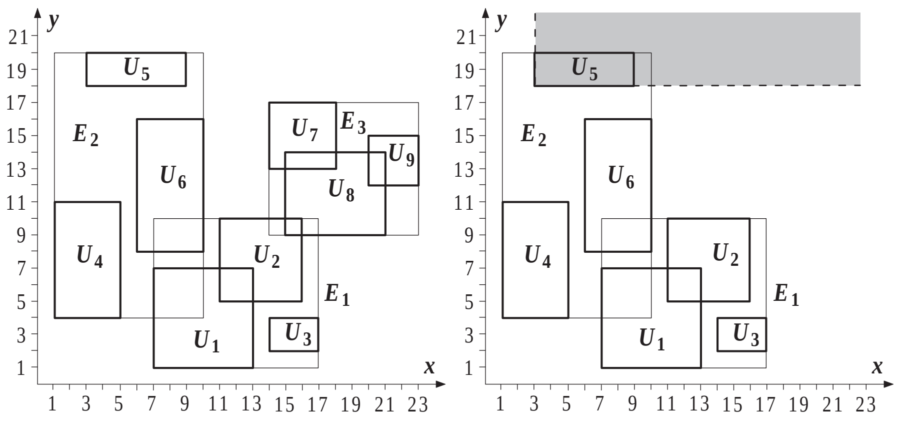
Let $U.MBB$ denote the minimum bounding box of the instances of an uncertain object U. U_{max} and U_{min} are the upper-right and lower-left corners of $U.MBB$, respectively. An object U is *fully dominated* by another object V if U_{min} is dominated by V_{max} or $U_{min} = V_{max}$ with the property that there is no instance from U allocated at U_{min} or there is no instance from V allocated at V_{max}. U is *partially dominated* by V if U_{max} is dominated by V_{min} but U is not fully dominated by V. Otherwise, U is *not dominated* by V. As depicted in Figure 3.4, U is fully dominated by V_1, partially dominated by V_2 and V_3, and is not dominated by V_4. Note that when U degenerates to one instance, the above concepts are immediately extendible if a point is treated as a special case of MBB.

An object is *redundant* if it is fully dominated by another object. Pei *et al* [PJLY07] shows the following theorem.

Theorem 3.1. *Regarding the discrete case, a redundant object U has 0 skyline probability, and any instance from another object V dominated by an instance of U is fully dominated by a non-redundant object, and thus has 0 skyline probability.*

Theorem 3.1 immediately applies that any redundant object can be immediately removed. This is because that any instance dominated by an instance from a redundant object must be fully dominated by another non-redundant object.

Weighted Centroids. Generally, the skyline probability of an object is deter-



(a) Set of Objects. (b) After Removing E_3.

Figure 3.5: Example.

mined by the distribution of its instances (*intra* relationships) and its relationships to the distributions of the instances of other objects (*inter* relationships). In our algorithms, we use the weighted (by probabilities) centroid $M(U)$ of instances to approximately represent the distribution of instances in object U to determine the processing order of objects. Formally, $M(U) = \sum_{u \in U} p_u \times u$.

R-**tree.** An initial computation in our algorithms is index-based, making use of the existing techniques. We assume that the minimum bounding boxes MBBs of objects' instances are indexed by an R-tree [Gut84]. A node of an R-tree contains a set of entries. Each entry in a leaf node is in the form <*obj*, *obj*.MBB> where *obj* refers to the object ID and *obj*.MBB is the MBB of the object's instances. Each entry in a non-leaf node has form <*child*, *child*.MBB> where *child* refers to a child node, and *child*.MBB is the minimum bounding box of this child node. Figure 3.5 (a) illustrates an R-tree built on MBBs of 9 uncertain objects. The root has 3 entries E_1, E_2, and E_3. Each child of the root encapsulate 3 objects, respectively.

BBS Algorithm. BBS algorithm [PTFS03] will be used and modified in the initial computation phase of our algorithms. Given a set of points indexed by an R-tree, BBS algorithm traverses an R-tree (built on points) to compute the skyline.

It maintains a *min-heap* H built against the *mindist* (minimum distance to the origin of the data space) of every entry (node). The algorithm goes iteratively. At the beginning, entries of the root are inserted into H. In each iteration, the top element e of H is processed. If e is fully dominated (i.e., the minimum corner of e is dominated) by an already computed skyline point, then e is discarded. Otherwise, if e is a data point, then it is output as a skyline point; if not, e is discarded and those entries of e which are not fully dominated by any already computed skyline point are inserted into H. An in-memory R-tree on already computed skyline points is maintained in order to facilitate examining the dominance relationship. The algorithm terminates when H is empty.

BBS ensures that each output point is in the skyline and it is I/O optimal.

3.2 Framework for TOPK-SOUND

Naively computing the skyline probability of each object is expensive and takes time $O(\sum_{\forall U,V} n_U \times n_V)$ for the discrete case where n_U and n_V are the number of instances in objects U and V, respectively. The computation regarding the continuous case may be even more expensive due to integrating PDFs.

In the light of efficiently computing top-k queries (i.e., pruning away none top-k objects as soon as possible), below we present a framework to efficiently support both exact computation and randomized computation. It consists of three Steps: *Preprocessing-Seeding-Final-Computation.*

Step 1: Preprocessing. Remove redundant objects.

Step 2: Seeding. Select k objects and compute their skyline probabilities.

Step 3: Final-Computation. Finalize the computation of top-k SOUND.

Note that after Step 1, if the number of objects left is not greater than k, then we can terminate the algorithm. Without loss of generality, we assume that there are more than k objects left after Step 1 in the rest of this section. Below, we present details in Step 1, Step 2, and Step 3.

3.2.1 Step 1: Preprocessing

As discussed in Section 3.1, Theorem 3.1 guarantees the correctness by removing redundant objects without affecting the skyline probability computation of remaining uncertain objects in discrete cases. Theorem 3.1 can be immediately extended to cover continuous cases. Note that if $u \prec v$, there must exist two small regions r_u and r_v surrounding u and v, respectively, such that r_u fully dominates r_v.

Theorem 3.2. *In continuous cases, a redundant object U also has 0 skyline probability. Moreover, any region in an uncertain object V fully dominated by a region in U is fully dominated by a non-redundant object, and thus has 0 skyline probability.*

Theorem 3.2 can be immediately verified according to the definitions. It implies that we can also remove the redundant objects regarding continuous cases without affecting the skyline probability computation for remaining objects.

We modify the original BBS algorithm to conduct the preprocessing. Below are the details of our modified BBS algorithm.

- An R-tree is built on MBBs; that is the unit data are MBBs of objects instead of points in the original BBS.
- Replace the dominance relationships among points by the *fully dominance relationship* among MBBs of objects in the modified BBS..
- In modified BBS, for every data entry in the R-tree we adopt the distance $d_M(U)$ between the centroid $M(U)$ and the origin as *mindist*, while for an

internal node (entry) in the R-tree, the minimum of such distances among the objects (data entries) contained is used as *mindist*; we assume that they are recorded in the R-tree.

It can be immediately verified that for two objects U and V, if $d_M(U) < d_M(V)$ then U will not be fully dominated by V. This guarantees no false positive and thus the modified BBS algorithm can remove all redundant objects.

Regarding the objects in Figure 3.5(a), after step 1, only the objects in Figure 3.5(b) remain.

3.2.2 Step 2: Seeding

The aim is to initially choose k objects with large skyline probabilities as a threshold to quickly prune away objects with small skyline probabilities without conducting the entire computation of their skyline probabilities. Intuitively, an object U with $M(U)$ as a skyline point, of all weighted centroids, may have more instances not to be dominated by other objects' instances; thus, it has a good chance to have a high skyline probability value. Moreover, we also give the preference to $M(U)$ with the smallest $d_M(U)$ since intuitively, smaller $d_M(U)$, less chance U being dominated by others.

Algorithm 3.1 Seeding

Choose the k objects U such that:

- $M(U)$ are skyline points with smallest $d_M(U)$, and
- if there are not enough skyline points $M(U)$, then choose the remaining objects V with the smallest $d_M(V)$ to make total k objects.

Algorithm 3.1 involves the computation of the skyline of the weighted centroids of non-redundant objects. This could be separately conducted after obtaining all non-redundant objects. In our algorithm, we conduct this simultaneously by executing the original BBS on centroids while computing non-redundant objects to share the costs since in our modified BBS $d_M(U)$ ($\forall U$) is already used as *mindist*; that is, we only need to maintain one heap.

Note that BBS, so does the modified BBS, always generates objects sorted increasingly on the search key *mindist* as a by-product. Thus, after running the modified BBS on objects and the original BBS on weighted centroids, the non-redundant objects U are sorted as a sorted list L_2 on $d_M(U)$ (non-decreasingly) and the objects with $M(U)$ as skyline points are also sorted as a sorted list L_1 against $d_M(U)$ (non-decreasingly). Clearly, L_2 contains all objects in L_1. Nevertheless, we can remove from L_2 the objects in L_1, while running BBS on the centroids and modified BBS simultaneously, as the objects are processed in the same order in these two algorithms. Algorithm 3.1 is thus executed in linear time $O(k)$.

Computing the skyline probability. The last phase of Step 2 (seeding) is to compute skyline probability for each seeded object, totally k objects. This will be conducted differently in the exact algorithm and the randomized algorithm.

3.2.3 Step 3: Final-Computation

The framework regarding this Step is outlined below. We iteratively process each object U as follows.

Filtering. If $P_{sky}(U) \leq \mathcal{P}_k$, then U is not a candidate of top-k SOUND. Here, $\mathcal{P}_k$ is the smallest skyline probability of the current top-k objects.

Refinement. Otherwise replace by U the object V, with the skyline probability $\mathcal{P}_k$, in the current top-k objects. Update $\mathcal{P}_k$.

Suppose that for each object U, $PD(U)$ denotes the set of objects each of which partially dominates U. Clearly, computing $P_{sky}(U)$ takes time $(n_U \times (\sum_{V \in PD(U)} |n_V|)$ if $PD(U)$ is already obtained.

An object is a *master object* if in the final-computation, it is processed to determine its current candidature of the result of top-k SOUND. The objects in $PD(U)$, when U is processed as a master object, are called *associated objects* to U. There are 2 key issues.

1. By which order are the associated objects (i.e., objects in $PD(U)$) accessed against a master object?

2. By which order are objects accessed as master objects?

Our experiments demonstrate that a random selection towards these two issues leads to the computation time an order of magnitude slower than the techniques developed below.

Order of Associated Objects. The main goal of this step is to develop efficient and effective techniques to prune away a master object U as earlier as possible; that is, access as less as possible the objects in $PD(U)$. The following theorem is immediate from the definitions.

Theorem 3.3. *Suppose that $\mathcal{U}'$ is a subset of the set $\mathcal{U}$ of objects. Then, $P_{sky}(U)|_{\mathcal{U}'} \geq P_{sky}(U)|_{\mathcal{U}}$, where $P_{sky}(U)|_{\mathcal{U}}$ ($P_{sky}(U)|_{\mathcal{U}'}$) denotes the skyline probability of U regarding the set $\{U\} \cup \mathcal{U}$ ($\{U\} \cup \mathcal{U}'$) of objects.*

The monotonic property in Theorem 3.3 implies that we only need to access objects $\mathcal{U}'$ if $P_{sky}(U)|_{\mathcal{U}'} \leq \mathcal{P}_k$.

Pruning Rule 3.1. *For an object U, let $\mathcal{U}$ be a subset of $PD(U)$. U can be excluded from the candidates of Top-k SOUND if $P_{sky}(U)|_{\mathcal{U}} \leq \mathcal{P}_k$.*

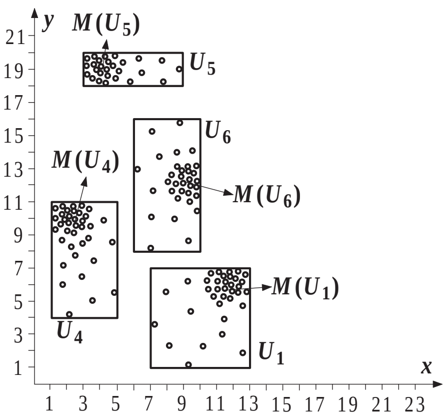


Figure 3.6: Data Distributions

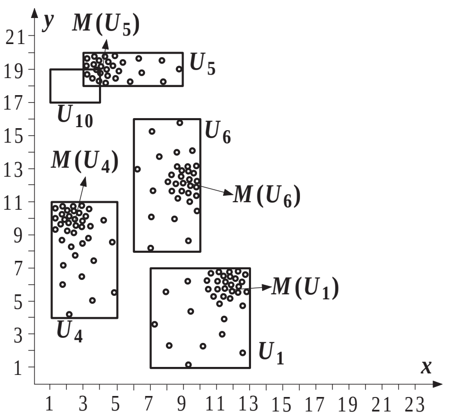


Figure 3.7: Dealing I_U

Example 3.3. *Regarding the example in Figure 3.5(b), suppose that* $k = 2$ *and objects* U_1 *and* U_4 *are initially chosen. When computing the probability value of* U_5*, we may find that the* $P_{sky}(U_5)|_{U_4}$ *is already smaller than* $\mathcal{P}_k$*. Thus, we no longer need to do a further computation between* U_5 *and* U_6*, nor* U_5 *and* U_1*.*

Continuing this example, suppose that the distributions of instances in U_1, U_5, U_4, and U_6, respectively, are as what illustrated in Figure 3.6. Clearly, it is better to start with the pair of U_4 and U_5 as the $P_{sky}(U_5)|_{U_4}$ is intuitively smaller than $P_{sky}(U_5)|_{U_1}$, or $P_{sky}(U_5)|_{U_6}$. Thus, there is a great chance to eliminate U_5 by computing $P_{sky}(U_5)|_{U_4}$ only.

Ideally we would like to find a *perfect* permutation, $\{U_i : 1 \leq i \leq |PD(U)|\}$, of the objects in $PD(U)$ such that $P_{sky}(U)|_{\{U_1,\ldots,U_i\}}$ is minimum among any subset of $PD(U)$ with i objects for each $1 \leq i \leq |PD(U)|$. That is, when U is pruned away from the candidates, the number of objects accessed from $PD(U)$ is always minimized. Nevertheless, such a permutation is not always possible.

Example 3.4. *For example, suppose that there are* 4 *objects* U*,* U_1*,* U_2*, and* U_3*, where* U *is the master object and has only two instances* u_1 *and* u_2 *with the equal probability* 0.5 *such that:*

- *the probabilities that u_1 is not dominated by U_1, U_2, and U_3 are* 0.15, 0.3, *and* 0.4, *respectively;*
- *the probabilities that u_2 is not dominated by U_1, U_2, and U_3 are* 0.9, 0.8, *and* 0.7, *respectively.*

It can be immediately verified that to follow the property specified above for each i in a perfect purnutation, the first associated object has to be accessed is U_1. Nevertheless when $i = 2$, the skyline probability of U against U_2 and U_3 is the minimum. Therefore, such a perfect permutation does not exist.

Below we develop a ranking function in (3.4) to order associated objects so that a good permutation may be obtained. It is based on the following observation. Let U and V be two uncertain objects where U is a master object. Consider that $M(U)$ approximately represents the instance distribution of object U. Intuitively, when $M(U)$ is dominated by an $M(V)$, we may expect that U has more chances to be dominated by V; in this case, the further the distance between $M(U)$ and $M(V)$ is, the larger the chance that the probability of U dominated by V has a larger value (i.e., the smaller the skyline probability of U).

$$\Delta(U, V) = \delta(U, V)d(M(U), M(V)).$$

Here, d is a distance metric; Manhattan distance is used in our implementation; $\delta(U, V) = 1$ if $M(V) \prec M(U)$, otherwise -1. Based on the above observations, in our algorithm we choose associated objects one-by-one increasingly according to Δ values.

Order of Master Objects. The goal is to make $\mathcal{P}_k$ increase as quickly as possible to reach the k-th largest skyline probability. Sharing with the same intuition in seeding algorithm (Algorithm 3.1), we choose a master object according to the priority described in Algorithm 3.1.

Algorithm. We present our algorithm to determine the final Top-k according to the above discussions.

Algorithm 3.2 Final-Computation

Input: L_1: a sorted list of remaining objects (with the weighted centroids as skyline points) on $d_M(U)$;

L_2: a sorted list of remaining objects (with the weighted centroids as non-skyline points) on $d_M(U)$;

Output: TOP_k: min-heap on the skyline probabilities of k objects, together with the corresponding object IDs;

Description:

```
1: for each $U$ of initial $k$ objects do $TOP_k$.push ($U$);
2: $\mathcal{P}_k := TOP_k.top.key$;
3: for each $U \in L_1 \cup L_2$ do
4:     if $Prob\ B\ (U, \mathcal{U}_{NR}) > \mathcal{P}_k$ then
5:         $TOP_k.pop()$; $TOP_k.push(U)$;
6:         $\mathcal{P}_k := TOP_k.top.key$;
7:     end if
8: end for
9: return $TOP_k$
```

As described in the seeding phase (Section 3.2.2), L_1 is a by-product of BBS algorithm, while L_2 is a by-product of the modified BBS. In Algorithm 3.2, $\mathcal{U}_{NR}$ is the set of non-redundant objects from a given set $\mathcal{U}$ of objects. We accessing $L_1 \cup L_2$ by firstly accessing L_1 and then accessing L_2 according to their sorted order, respectively.

To save the storage space, in TOP_k we only keep object IDs and their corresponding skyline probabilities. The method $Prob\ B\ (U, \mathcal{U}_{NR})$ checks the candidature of U and then (possibly) calculates the skyline probability of object U; this will be conducted differently in the exact algorithm and the randomized algorithm with different pruning techniques; details will be presented in the next two sections.

3.3 Exactly Computing TOPK-SOUND

The exact algorithm for discrete cases follows the framework of 3 steps in Section 3.2. In this section, we present the details of computing the skyline probability in Step 2 - Section 3.2.2, and our pruning strategies in Step 3 - Section 3.2.3.

Computing the skyline probability. The last part of Step 2 computes the skyline probabilities of the initially chosen k objects.

Note that an in-memory R-tree on MBBs of non-redundant objects has been built as a by-product of our modified BBS, corresponding to the in-memory R-tree on skyline points in the original BBS.

For each U of these initially chosen k objects, our algorithm to compute its skyline probability is conducted in two stages. At stage 1, it iteratively traverses the in-memory R-tree in a depth-first manner to search for objects with MBBs partially dominating U; that is, search for objects in $PD(U)$. Once such an object V is found, it performs an update of the skyline probability of each instance of U. In our implementation, the *Synchronous Traversal* (ST) join paradigm $ST(U, V)$ [HJR97] is adopted instead of trivially comparing each pair of instances from U and V. Let CH_R denote the set of children of the root of R. Algorithm 3.3 presents our algorithm to update the skyline probability. Note that there are only two relationships among non-redundant objects: partially dominating or not dominating. If R' does not (partially) dominate U, then R' can be simply passed-over since each $u \in U$ has the probability 1 not being dominated by any object in R'.

Algorithm 3.3 $Prob(U, R)$

Input: R: an in-memory R-tree index of non-redundant objects;

U: an object;

Output: $P_{sky}(U)$

Description:

```
1: $Q := CH_R$;
2: remove entries from $Q$ that do not (partially) dominate $U$;
3: **While** $\{Q \neq \emptyset\}$
4:     $R' := Q.pop()$;
5:     **if** $\{R'.MBB$ does not (partially) dominate $U\}$ **then**
6:         pass-over $R'$;
7:     **else**
8:         **if** $\{R'$ is an object $V \neq U\}$ **then**
9:             ST(U, V);
10:            $U := REMOVE\_ZERO(U)$;
11:        **else** $Q := Q \cup CH_{R'} - \{R'\}$;
12:    $P_{sky}(U) := \sum_{u \in U} p_u \times P_{sky}(u)$;
13: **end while**
14: **return** $P_{sky}(U)$
```

In Algorithm 3.3, Q is maintained as a queue. To facilitate the synchronous traversal join paradigm, the instances in each object are pre-organized by an in-memory R-tree data structure such that at each node E, we also record p_E - the summation of the probabilities of the instances (to appear) in E. $ST(U, V)$ is a simple modification of the synchronous traversal join algorithm to conduct an in-memory update of the skyline probability of each instance in U due to an addition of object V. We only need to modify the join condition to "one rectangle (point) fully dominates another rectangle (point)". In $ST(U, V)$, for a pair of node $E \in U$

and node $E' \in V$, there are 3 cases below.

Case 1: If E' does not dominate E, pass-over E'.

Case 2: If E' fully dominates E, then $Pr(E) := Pr(E) + p_{E'}$. Note that $Pr(E)$ is initiated to 0 when a new object is added, and is the summation of the occurrence probabilities of instances in an E' which is captured in $ST(U, V)$ to fully dominate E.

Case 3: Otherwise (E' partially dominates E), put (not Case 1) pairs of children of E' and E in a queue for further traversal.

To compute the skyline probability correctly, after performing $ST(U, V)$ for an object V, we push down $Pr(E)$ from each internal node E to the leaf nodes (instances) along the tree path. That is, $Pr(u) = \sum_{E \in l_u} Pr(E)$ where l_u is the path from the root to the leaf u. Note that the whole push-down computation can be performed in linear time if it is executed in a top-down fashion. Moreover, after push-down, we update $P_{sky}(u)$ to $P_{sky}(u) := P_{sky}(u)(1 - Pr(u))^2$, and then reset $Pr(E) = 0$ for each entry E in the R-tree, including leafs.

Remove Instances with Skyline Probability 0 by REMOVE_ZERO (U). The following Theorem is fundamental.

Theorem 3.4. *Suppose that an instance u has 0 skyline probability. Then, there must be a non-redundant object V such that u is fully dominated by V; that is, each instance of V fully dominates u. Moreover, no instance from V has skyline probability 0.*

Proof. Suppose that PS is the set of objects such that the right-upper corners of all objects in $U \in PS$ form the skyline against all those of the non-redundant objects.

[2]Note that $P_{sky}(u)$ is initialized to 1.

It can be immediately verified that one object from PS must fully dominate u. Moreover, it is also immediate that none of instances in an object U in PS has 0 skyline probability. □

Instances with skyline probability 0 can be removed from U from further considerations. Firstly, removing them from U implies that while computing the skyline probability of U, these instances will not be counted. This is equivalent to counting their probabilities as 0. Secondly, removing them from U will not affect the computation of skyline probabilities of other objects. This is because any instance v fully dominated by an instance u with skyline probability 0 must be fully dominated by a non-redundant object V' without any instance removed according to Theorem 3.4. Consequently, the 0 skyline probability can be discovered from the relationship between V' and u. Thirdly, removing these instances not only saves the memory space for the scalability but also reduces the computation costs when U is used in computing the skyline probabilities of other objects.

Example 3.5. *Regarding the example in Figure 3.6, once the instances in* U_6 *with skyline probability* 0 *are removed, we no longer need to use them when update the skyline probability of* U_5 *by adding* U_6*.*

In $REMOVE\_ZERO(U)$, we remove the instances with 0 skyline probability; if an entry in the R-tree on U's instances only contains the instances with 0 skyline probability, then the entry is removed as well. We do not re-balance the R-tree of U as our initial experiment demonstrates that such re-balance costs cannot be paid off. Note that we do not physically remove instances or entries from a pre-built in-memory R-tree; instead, we mark out those "removed" instances and entries to prevent them from being involved in further computation.

Processing *Prob B* $(U, \mathcal{U}_{NR})$**.** It can be done exactly in the same way as Algorithm

3.3. However, as implied by Theorem 3.3, we do not have to always conduct the entire computation of $P_{sky}(U)|_{\mathcal{U}_{NR}}$. To facilitate this, we always choose an entry with the largest Δ value. As discussed above, $\mathcal{U}_{NR}$ is the set of non-redundant objects that are indexed by an in-memory R-tree as a by-product of our modified algorithm.

It can be immediately verified that Theorem 3.3 also holds for the situation where instances with skyline probability 0 are removed; this together with Theorem 3.3 yields that we can add associated objects one-by-one to calculate skyline probability and prune away a master object U against a subset of $PD(U)$ based on Pruning Rule 3.1.

Prob B (U,$\mathcal{U}_{NR}$) modifies *Prob*(U, R) (Algorithm 3.3) as follows.

- In *Prob B* (U,$\mathcal{U}_{NR}$), we maintain a max-heap Q based on Δ values of the R-tree entries instead of a queue where Δ values are calculated on the fly. To retain the monotonic property that for each internal entry E, its Δ value is the maximum of Δ values of the entries/objects contained,

  > for each entry E, we record the lower-left corner $u_{E,M}$ of the minimum bounding box of the weighted centroids of objects contained in E; then compute the Δ value using $M(U)$ and $u_{E,M}$.

- Between lines 10 and 11 in Algorithm 3.3, we calculate the current skyline probability after adding one associated object; that is, add $\sum_{u \in U} p_u P_{sky}(u)$. If it is already not larger than $\mathcal{P}_k$, then we terminate *Prob B* (U, R); consequently U is excluded from the candidates of top-k SOUND; that is, the condition in line 4 of Algorithm 3.2 does not hold.

Remark. A non-redundant object may have 0 skyline probability for every instance. As illustrated in Figure 3.8, U is a non-redundant object. Nevertheless,

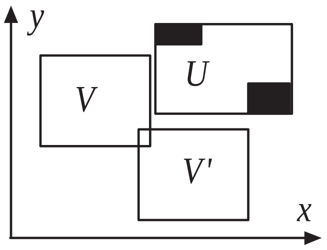


Figure 3.8: Multiple Dominance Relationships.

the skyline probability of U is zero if its instances are located in the two black-colored boxes only.

Generally, an object U may have a subset I_U of instances such that each instance in I_U has 0 skyline probability. I_U can be removed from U without further consideration, as explained before. This can be done by using window query techniques to detect the instances dominated by the skyline points on the upper-right corners of MBBs of non-redundant objects. However, our initial experiments suggest that such pre-computation costs cannot be paid off.

In our algorithm, we only remove a subset of instances, captured with 0 skyline probability on the fly, in examining its top-k candidature. For instance, regarding the example in Figure 3.7, U_5 may be excluded from a top-k candidate after computing $Psky(U_5)_{U_4}$ without examining U_{10}. Consequently, before U_5 is excluded from a further consideration (i.e. examining U_{10} etc.) as a master object we are only able to capture the set of instances in U_5 fully dominated by U_4 but not those fully dominated by U_{10} only.

3.4 Randomized Algorithm

In this section, we present a randomized algorithm to deal with both continuous (with the assumption that PDFs are *continuous* functions) and discrete cases. Let

$\mathcal{U} = \{U_j | 1 \leq j \leq n\}$ be the set of non-redundant objects. The basic idea is to sample all possible worlds, $\prod_{i=1}^{n} U_j$, by m possible worlds S_i; that is, each sample S_i ($1 \leq i \leq m$) consists of n randomly chosen points for the continuous case (or instances for the discrete case) - one per each object. Then, we use $\frac{m_U}{m}$ as the approximation of the skyline probability of an object U to determine the solution for top-k SOUND. Here, m_U is the number of times that object U is involved as the skyline points in these m samples (worlds).

Example 3.6. *Consider the example in Figure 3.3. Regarding the two samples (worlds) (i.e., $m = 2$) (a_1, b_1, c_2) and (a_2, b_1, c_1), $m_B = 1$.*

Algorithm 3.4 Randomized Algorithm

Input: $\mathcal{U} = \{U_i | 1 \leq i \leq n\}$; m: an integer.

Output: T_k: k objects.

Description:

```
1: for $i := 1$ to $m$ do
2:     for $j := 1$ to $n$ do
3:         $u_{i,j} := random(U_j)$;
4:     end for
5: end for
6: Sky-COUNT ($\{u_{i,j} | 1 \leq i \leq m, 1 \leq j \leq n\}$);
7: $T_K :=$ the $k$ objects $U$ with the largest $\frac{m_U}{m}$;
8: return $T_k$
```

In Algorithm 3.4, regarding the discrete case we use $random(U_j)$ to randomly select an instance from U_j such that each instance $u \in U_j$ has the probability p_u to be selected. Regarding a continuous case, we first divide the whole data space, the MBB of U, into very small regions such that in each region, the difference of values of a PDF is bounded by a very small value ξ. Then, $random(U_j)$ randomly selects a point from a region r with probability $Pr(r)$. Sky-COUNT ($\{u_{i,j} | 1 \leq i \leq m, 1 \leq j \leq n\}$) computes m_U for each object U. Computing the skyline of each sample $S_i = \{u_{i,j} | 1 \leq j \leq n\}$ (for $1 \leq i \leq m$) to get each m_U is expensive, even more expensive than the exact algorithm, when m is reasonably large. Below, we will present an efficient counting technique. First, we present the accuracy guarantee of Algorithm 3.4.

3.4.1 Accuracy Guarantee

For each object U_j, in Algorithm 3.4 the events whether $u_{i,j} = random(U_j)$ is a skyline point of S_i is described by the following totally independent random variable.

$$X_{i,j} = \begin{cases} 1 & \text{if } u_{i,j} \text{ is a skyline in sample } i \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

It is immediate that $E(X_{i,j}) = \sum_{u \in U_j} p_u P_{sky}(u)$ in a discrete case for each i and j. For a continuous case, without loss of generality we may assume that for each object U its PDF domain is a finite region and S is the maximum of the region volumes of these n uncertain objects.[3] We choose ξ such that $\xi < \frac{\epsilon}{2^n \times S \times 4}$. Consequently, it is immediate that $E(X_{i,j}) = \int_{u \in U_j} f_{U_j}(u) \prod_{V \neq U_j} (1 - \int_{v \prec u, v \in V} f_V(v) dv) du) + \xi_{U_j}$ where for each U, $0 \leq |\xi_U| < \frac{\epsilon}{4}$; that is, $E(X_{i,j}) = P_{sky}(U_j) + \xi_{U_j}$. Let

$$X_j = \frac{\sum_{i=1}^{m} X_{i,j}}{m} \tag{3.5}$$

We have $\frac{m_{U_j}}{m} = X_j$ and $E(X_j) = P_{sky}(U_j) + \xi''_{U_j}$ where $|\xi''_{U_j}| \leq \frac{\epsilon}{8}$. Given a set of objects, for $1 \leq l \leq k$ let P_l denote the skyline probability of the object with the

[3]For a PDF of an uncertain object U with a infinite domain, we can simply cut the infinite part of the domain with a very small probability ξ'_U. Then, the following analysis still holds.

l-th largest skyline probability. Suppose that for $1 \le l \le k$, U_{j_l} is ranked as the top l-th object by Algorithm 3.4. Note that the object U_{j_l} could be different than the real top l-th object (with skyline probability P_l). Nevertheless, the following theorem states that when the sample size m is sufficiently large, X_{j_l} $(= \frac{m_{U_{j_l}}}{m})$ will be an ϵ-approximation of P_l with confidence $1 - \delta$.

Theorem 3.5. *Given an ϵ ($0 < \epsilon < 1$), a δ ($0 < \delta < 1$), and n non-redundant objects, if $m = O(\frac{1}{\epsilon^2} \log \frac{n}{\delta})$ and ξ is chosen with $\xi < \frac{\epsilon}{8 \times 2^n \times S}$, then*

$$Pr(\wedge_{l=1}^{k} \{|X_{j_l} - P_l| \le \epsilon\}) \ge 1 - \delta.$$

To prove Theorem 3.5, we need the following Lemmas. By the **Chernoff/Hoeffding** bound (Theorem 2.7 in [Gol]), the following Lemma is immediate. It implies that when the sample size is sufficiently large, X_j is very close to the skyline probability of U_j with a high confidence.

Lemma 3.1. *$Pr(|X_j - P_{sky}(U_j)| \ge \lambda) \le 2exp^{-2m(\lambda - \frac{\epsilon}{8})^2}$ ($0 < \lambda \le 1$) $\forall j \in [1, n]$.*

The following lemma states that if the sample size is sufficiently large, then with a high confidence, Algorithm 3.4 can only reverse the order, against their skyline probabilities, of two objects with a small difference between their skyline probabilities.

Lemma 3.2. *For j and j', suppose that $P_{sky}(U_j) < P_{sky}(U_{j'})$. Then, $Pr(X_j \ge X_{j'}) \le exp^{-m(P_{sky}(U_j') - P_{sky}(U_j) - \frac{\epsilon}{4})^2/2}$.*

Proof. Let $Z = X_j - X_{j'}$. We have $Pr(X_j \ge X_{j'}) \le Pr(Z - E(Z) > P_{sky}(U_{j'}) - P_{sky}(U_j) - \frac{\epsilon}{4})$. By Hoeffding inequality (Theorem 2 in [Hoe63]), the lemma is immediate. □

Proof of Theorem 3.5 Without loss of generality, suppose that for an $l \in [1, k]$, the object U_l has the l-th largest skyline probability P_l (i.e. $P_{sky}(U_l) = P_l$). For each object U_{j_l} (ranked the l-th by Algorithm 3.4), we prove the probability of the following 3 events to appear is small when m is chosen appropriately.

Event 1: $P_{sky}(U_l) - P_{sky}(U_{j_l}) > \epsilon/2$.

Event 2: $P_{sky}(U_{j_l}) - P_{sky}(U_l) > \epsilon/2$.

Event 3: $|X_{j_l} - P_{sky}(U_{j_l})| > \epsilon/2$.

Let $m = \frac{8}{\epsilon^2} \log \frac{2(n+1)k^2}{\delta}$.

If Event 1 occurs, then $\exists U_i$ such that $P_{sky}(U_i) \geq P_l$ (i.e. $i \leq l$) and $X_i \leq X_{j_l}$. This implies that U_i and U_{j_l} change their order by Algorithm 3.4. Consider that there are l such objects. Consequently, from Lemma 3.2, the total probabilities that U_{j_l} change the order with these l objects is bounded by $\frac{l\delta}{4k^2(n+1)}$. Therefore, $Pr(Event\ 1) \leq \frac{l\delta}{4k^2(n+1)}$.

If Event 2 occurs, then $\exists U_i$ such that $P_{sky}(U_i) \leq P_l$ (i.e. $i \geq l$) and $X_i \geq X_{j_l}$. Note that there are $(n-l+1)$ such objects. Similarly, from Lemma 3.2, we have $Pr(Event\ 2) \leq \frac{(n-l+1)\delta}{4k^2(n+1)}$.

Let $\lambda = \epsilon/2$. By Lemma 3.1, $Pr(Event\ 3) < \frac{\delta}{2k}$.

Therefore, for all l ($1 \leq l \leq k$) the probability of that one of these $3k$ events occurs is not greater than δ. Consequently, the theorem holds since $\frac{16}{\epsilon^2} \log \frac{2(n+1)k^2}{\delta} = O(\frac{1}{\epsilon^2} \log \frac{n}{\delta})$.

Discussions. Note that the sample size in Theorem 3.5 is irrelevant to the number of instances in an object. If we run the seeding phase, and choose ϵ as $\epsilon_1 \mathcal{P}_k$ if $\mathcal{P}_k \neq 0$, then we can guarantee a relative ϵ-approximation theoretically. Theoretically, to guarantee ϵ-approximation, we need a sample size as stated in Theorem 3.5; nevertheless, our experiment demonstrate that $m = 1000$ can provide a quite accurate solution to top-k SOUND.

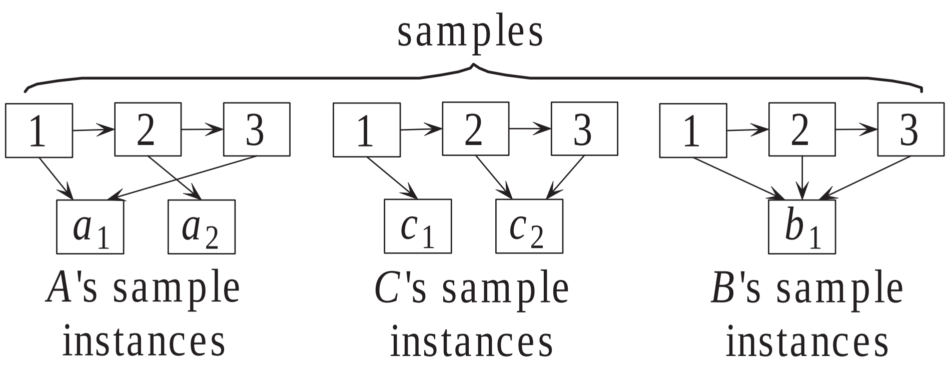


Figure 3.9: Example of Samples.

3.4.2 Efficient Algorithm

The Sky-COUNT in Algorithm 3.4 follows the framework in Section 3.2; that is, 3 steps: preprocessing, seeding, and final computation. While others are exactly the same as those in the exact algorithm, we present efficient techniques to compute $\frac{m_U}{m}$ (corresponding to Algorithm 3.3 - *Prob*()) for the initially chosen k objects in the seeding phase and execute *Prob B* () in Algorithm 3.2 in Step 3. We aim to directly compute m_U for each object U by avoiding computing the skyline for each sample.

In our techniques, we organize sampled instances (points) of each object U as a list $U.list$ to save the storage space by removing duplicates.[4] Initially, the i-th node of this linked list contains the integer i, and the reference referring to the instance of U in the i-th sample. For instance, regarding the example in Figure 3.3, 3 samples are dropped, $S_1 = \{a_1, c_1, b_1\}$, $S_2 = \{a_2, c_2, b_1\}$, and $S_3 = \{a_1, c_2, b_1\}$. Their linked lists are illustrated in Figure 3.9.

The basic idea of our counting algorithm is as follows. If a sampled instance (point) u of an object U is dominated by a sampled instance (point) of another object V from the same sample, we simply remove the sampled instance u from the

[4]An instance may appear in several samples especially in the discrete case.

linked list. In the end, the number of sampled instances remained in each object U is m_U. Regarding the above example in Figures 3.3 and 3.9, in *A.list* three sampled instances $\{a_1, a_2, a_1\}$ are left after our algorithm; thus $m_A = 3$. In *C.list* and *B.list*, 0 and 2 sampled instances ($\{b_1, b_1\}$) are left, respectively; thus $m_C = 0$ and $m_B = 2$.

While our counting techniques follow the framework of EXACT algorithm except that $Prob(U, V)$ and $Prob\ B\ (U, V)$ are executed differently. We do not use a tree-like data structure to organize the sampled instances (points). Consequently, we replace the $ST(U, V)$ in both $Prob(U, V)$ and $Prob\ B\ (U, V)$ by *dominance check* $DC(U, V)$ which checks the dominance relationship between the instances of U and V within the same sample.

In $DC(U, V)$ where U is a master object, we adopt the same traversal strategy as the sort-merge join between *U.list* and *V.list* since they are sorted on sample subindexes. Once u and v are found in the same sample (by their corresponding sample subindexes), we remove the sampled instance (point) from *U.list* if $v \prec u$, or remove the sampled instance (point) from *V.list* if $u \prec v$. An instance (point) is removed if there is no sampled instance (point) referring to it any more.

Example 3.7. *In the example in Figure 3.9, in $DC(C, A)$ after checking against the first sample index* 1*, element* 1 *is removed from C.list. Since instance c_1 only has element* 1 *referring to it, instance c_1 is also removed.*

Clearly, the complexity of $DC(U, V)$ is $O(dm)$ where d is the dimensionality. Moreover, Sky-COUNT runs in time $O(d \times n \times m \times a + dn\mathcal{F}(n))$. Here, a is the average number of associated objects to a master object and $\mathcal{F}(n)$ represents the average costs for one master object to obtain associated objects. Consequently, if a is a constant and m is a constant (say, 1000), then the time complexity of our Sky-COUNT is $O(dn\mathcal{F}(n))$. While there is no theoretical guarantee on $\mathcal{F}(n)$

regarding an R-tree, we could expect that $\mathcal{F}(n)$ is poly-log (n) in practice in a low dimensional space when a is a constant.

3.5 Computing p-Skyline

Our exact and randomized algorithms can be immediately extended to compute p-skyline proposed in [PJLY07]; that is, for a given threshold p $(0 \leq p \leq 1)$, compute all uncertain objects U such that $P_{sky}(U) \geq p$. Below are the modifications.

Regarding the framework in Section 3.2, in our exact and randomized algorithms for computing p-skyline we keep Step 1 but do not use Step 2. In Step 3 of both algorithms, we prune away the objects U with skyline probabilities (or $\frac{m_U}{m}$) below a pre-given threshold p, the Step 3 of both exact and randomized algorithms can be immediately applied to compute the p-skyline. The modified algorithms are named p-EXACT and p-RAND, respectively.

It can be immediately verified that the p-EXACT is correct. Moreover, by the Chernoff/Hoeffding bound (Theorem 2.7 in [Gol]) together with the fact that for each output object U $\frac{m_U}{m} \geq p$, the following theorem regarding accuracy immediately holds.

Theorem 3.6. *Given an ϵ $(0 < \epsilon < 1)$ and a δ $(0 < \delta < 1)$, let $m = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ (the sample size in p-RAND). For each object U output by p-RAND, $Pr(P_{sky}(U) - p < -\epsilon) \geq 1 - \delta$, where p is a given probability threshold in the problem of p-skyline.*

Theorem 3.6 states that with confidence $1-\delta$, the objects output by the algorithm p-RAND with the skyline probability not less than $p-\epsilon$. Note that Theorem 3.6 immediately implies that if we replace $m = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ by $O(\frac{1}{(p\epsilon)^2} \log \frac{1}{\delta})$, then we will have an relative ϵ-approximation guarantee, that is, with confidence $1-\delta$, the objects output by the algorithm p-RAND with the skyline probability not less than $(1 - \epsilon)p$.

Following similar arguments to those in TOP-k SOUND, the algorithm p-RAND is immediately applicable to the continuous case with the above accuracy guarantee.

Beside the theoretical guarantee of accuracy as above, our experiment demonstrated that p-RAND has already been very accurate when m reaches 1000.

3.6 Performance Evaluation

We report an extensive empirical study to evaluate the effectiveness and the efficiency of our algorithms. All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel P4 2.8GHz CPU and 2G memory under Debian Linux.

The following algorithms are implemented and evaluated.

1. *EXACT* : the exact algorithm proposed in Sections 3.2 and 3.3.
2. *TEXACT* : the trivial version of the exact algorithm in which the order of accessing master objects and associated objects of a given master object is randomly conducted instead of being arranged as described in Sections 3.2 and 3.3.
3. *RAND* : the randomized algorithm proposed in Section 3.4.
4. *TRAND* : the trivial randomized algorithm using SFS algorithm [CGGL03] to compute the skyline of each sample.

Our experiments are conducted on the real dataset and synthetic datasets.

Real dataset. We use the NBA game-by-game technique statistics from 1991 to 2005. The NBA dataset is downloaded from `www.nba.com` and consists of 339,721 records (instances) of 1,313 players. We treat each player as an uncertain object and the records of a player as the instances of the corresponding object. Instances of an object take equal probability to appear. Three attributes are selected in our experiments: the number of points, the number of assistants, and the number of rebounds. The larger the attribute values, the better.

Synthetic datasets. We generate discrete synthetic datasets and continuous synthetic datasets where objects are represented by instances and PDFs, respectively.

For both kinds of datasets, the domain of each dimension is $[0, 1]$ and the dimensionality d varies from 2 to 5 with the default value 3. We first generate the centres of n uncertain objects using the benchmark data generator in [BKS01], where n varies from $10,000$ to $100,000$ with the default value $10,000$. Anti-correlated and Independent distributions of n object centres are used in our experiments. By default, we use Anti-correlated distribution. Then, for each uncertain object we create a hyper-rectangle region where the instances or the PDF of this object appear. The edge size of the hyper-rectangle region follows a Normal distribution in range $[0, 0.2]$ with expectation 0.1 and standard deviation 0.025.

For discrete synthetic datasets, the number of instances of an uncertain object follows a Uniform distribution in range $[1, h]$ where h varies from 400 to $5,000$ with the default value 600. In expectation, each object has $\frac{h}{2}$ instances and the total number of instances in a dataset is $\frac{hn}{2}$; by default, it is $3,000,000$. In our experiments, two largest datasets have total instances of $\frac{5,000}{2} \times 10,000 = 25,000,000$ and $\frac{600}{2} \times 100,000 = 30,000,000$, respectively.

Instances of an uncertain object follow a *Uniform* or *Zipf* distribution in our experiments. In Uniform distribution, the instances of an object are distributed uniformly in the region and have the same probability. In Zipf distribution, for an object firstly an instance u is randomly generated, the distances of other instances to u follow a Zipf distribution with $z = 0.5$; the probabilities of each generated instances also follow a Zipf distribution with $z = 0.2$.

Consider distributions of object centres and the instances within an object, we have Anti-Uniform datasets where the centers of object MBBs follow Anti-correlated distribution, while instances follow Uniform distribution. Similarly, we have Inde-Uniform, Anti-Uniform, Anti-Zipf, and Inde-Zipf datasets.

For continuous synthetic datasets, the PDF of each object is *Uniform* or *Constrained-Normal* (Con-Nor for short), while other settings are generated in the same way as the discrete case. For a Uniform distribution, the PDF of an object U is a constant. The Con-Nor distribution is a Normal distribution constrained within the region (i.e., MBB) of an object U given by the formula below,

$$pdf_{CN}(x) = \begin{cases} pdf_N(x)/\lambda & if x \in U \\ 0 & otherwise \end{cases} \tag{3.6}$$

Here, $\lambda = \int_{x \in U} pdf_N(x)\mathrm{d}x$ and the d dimensional pdf of Normal distribution $pdf_N(x) = \frac{1}{(\sqrt{2\pi\sigma^2})^d} e^{-\frac{1}{2\sigma^2}\sum_{i=1}^{d}(x.D_i - \mu.D_i)^2}$, where $\mu.D_i$ $(1 \le i \le d)$ is the coordinate of the centre of U on dimension D_i and the standard deviation $\sigma = 0.025$.

We use Anti-Uniform-PDF to denote continuous synthetic datasets with Anti-correlated distribution for the centers of object MBBs and Uniform PDF for each object. Similarly, we have Anti-Con-Nor-PDF datasets.

In our experiments, k varies from 10 to 100 with the default value 30. The sample size m varies from 1000 to 2500 with the default value 1000.

Note that *in our experiments all parameters use default values unless otherwise specified.* Table 6.2 summaries the experiment settings.

3.6.1 Evaluating Efficiency

In this subsection, we evaluate the efficiency of our algorithms on the NBA dataset and discrete synthetic datasets. Note that we only report the performance of

Notation	Definition (Default Values)
n	number of uncertain objects in the dataset (10K)
k	number of uncertain objects in SOUND (30)
d	dimensionality of the dataset (3)
h	largest number of instances in an objects (600)
m	number of samples in randomized algorithms (1000)
$\mathcal{D}$	dataset types (Anti-Uniform)

Table 3.2: Experiment Settings.

algorithm RAND against discrete synthetic datasets because for the same type of probability distribution of instances (e.g., instances follow Uniform distributions), RAND has almost the same performance for the discrete and continuous cases; consequently we only report the results for the discrete case.

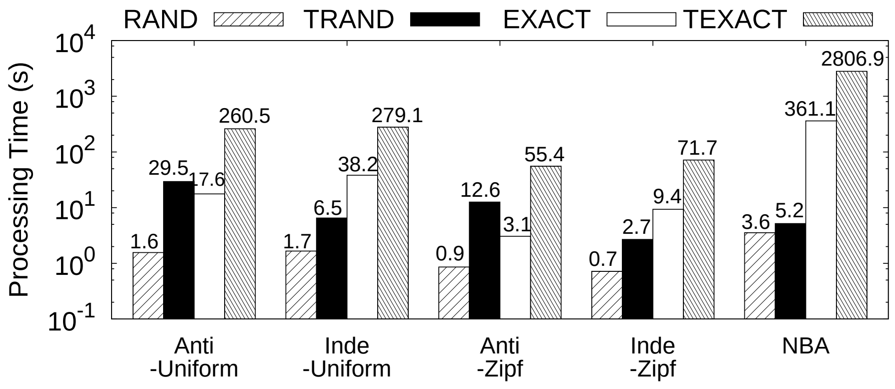


Figure 3.10: Different Datasets

Figure 3.10 shows the running time of the four algorithms on various datasets. While both EXACT and RAND are quite efficient, RAND is more efficient than EXACT on both synthetic and real datasets. EXACT performs poorly on the NBA dataset. This is because in the NBA dataset most objects are partially dominated by many others; thus many join-like operations (i.e., $ST(U,V)$) have been performed. In this case, the linear time complexity of $DC(U,V)$ of RAND shows a great advantage.

Figure 3.10 also demonstrate that TEXACT is upto 9 times slower than EXACT; this shows the great advantage of developed accessing order techniques regarding associated and master objects, respectively. RAND is also significantly more efficient than the trivial randomized algorithm TRAND. Therefore, in the remaining experiment part we can exclude the performance evaluation of TEXACT and TRAND since they provide the same results as EXACT and RAND, respectively, but are much more slower.

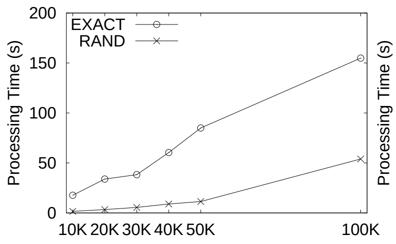


Figure 3.11: Varying n

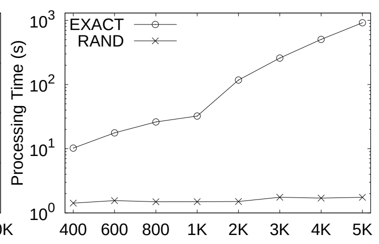


Figure 3.12: Varying h

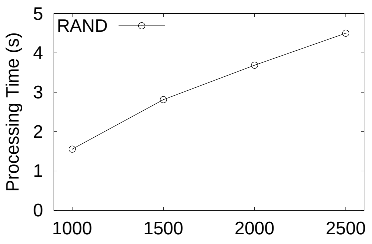


Figure 3.13: Varying m

The results of the second experiment reported in Figure 3.11 demonstrates that the performance of both EXACT and RAND are quite scalable when the number of uncertain objects increases.

The third experiment reported in Figure 3.12 shows that RAND is not sensitive to the number of instances (regarding the discrete case) since only a fixed number of samples are generated in randomized computation. On the other hand, the performance of EXACT drops when the number of instances grows.

Figure 3.13 shows that the running time of RAND increases linearly against the increment of the sample size m. This is because our counting technique runs in linear time with respect to m.

Figure 3.14 demonstrates the impact of k on performance. RAND is more efficient and the processing time grows much slower than EXACT because the cost for seeding the first k objects in RAND is not as dominating as that in EXACT.

Figure 3.15 evaluates the impact of dimensionality. It shows that the running time of EXACT significantly decreases when d increases. This is because when d is large, the dominating relationships among objects are weak.

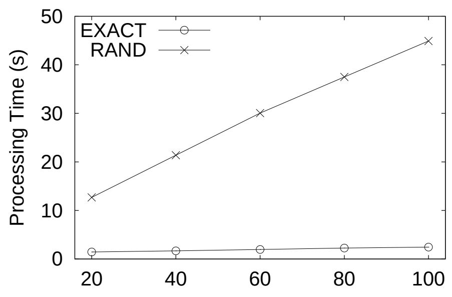


Figure 3.14: Varying k

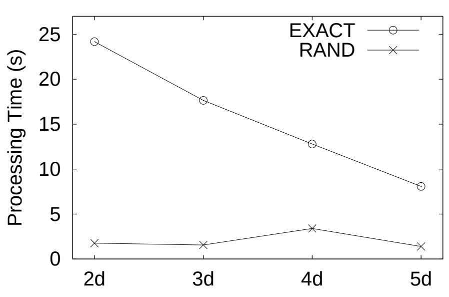


Figure 3.15: Varying d

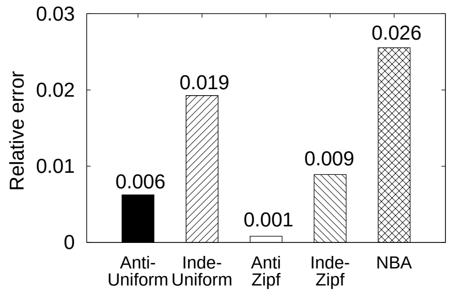


Figure 3.16: Different Datasets

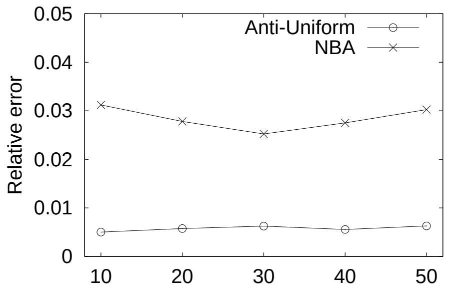


Figure 3.17: Varying k

3.6.2 Evaluating Accuracy

We evaluate the accuracy of RAND regarding both discrete and continuous cases. We use the average *relative errors*, the average of $\frac{|P_i' - P_i|}{P_i}$ (for $1 \leq i \leq k$), to measure the accuracy, where P_i is the i-th largest skyline probability and P_i' is the estimated skyline probability of the i-th element returned by RAND.

Discrete Cases

We first evaluate RAND on the NBA dataset and discrete synthetic datasets.

Figure 3.16 illustrates the accuracy of RAND on various datasets. It shows that $m = 1000$ already gives very accurate results (average relative error < 0.03). The NBA dataset has the worst performance; this is because that skyline probabilities are small due to a high overlapping degrees among MBBs of uncertain objects.

Figure 3.17 reports the impact of k on accuracy. It shows that when k is not large (typical situation for a top-k problem), the accuracy is not very sensitive to k.

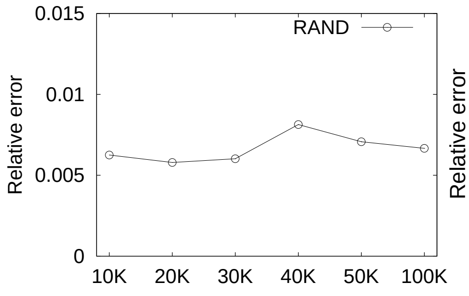


Figure 3.18: Varying n

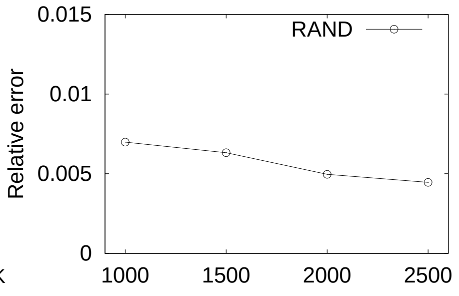


Figure 3.19: Varying m

Figure 3.18 shows that the accuracy is not quite related to the number of objects n.

Figure 3.19 shows that the accuracy increases (i.e., relative errors decrease) when the sample size grows, as what we expected.

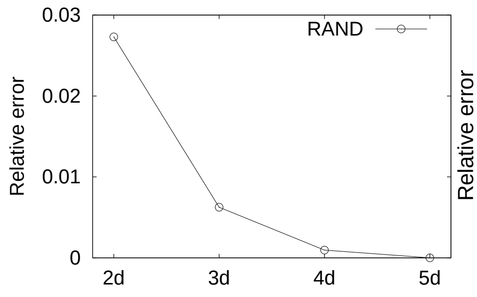


Figure 3.20: Various d

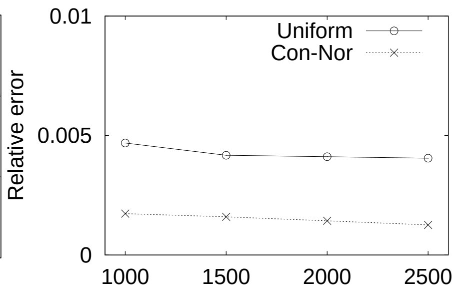


Figure 3.21: Various m

Figure 3.20 shows that the average relative error drops (i.e., accuracy increases) quickly as the dimensionality increases. This is because the average skyline probability of objects increases when the dimensionality increases.

Continuous Cases

RAND is also very accurate in the continuous case. We run RAND on continuous synthetic datasets with Anti-Uniform-PDF and Con-Nor-PDF, while other parameters adopt default values.

For datasets of Anti-Uniform-PDF, the skyline probability of an object is computed following Equation (3.3) where all integrals can be computed as the volumes of the corresponding regions. To generate samples in RAND, the coordinates of sampled points are generated uniformly within the MBB of an object.

For datasets of Anti-Con-Nor-PDF, since the integral of Gaussian function cannot be evaluated exactly, each object is discretized by drawing $10,000$ samples and then run the algorithm EXACT against the discretized objects to get the skyline probability of each object. To generate samples, the coordinates of sampled points are generated according to the Con-Nor distribution of each object. We use GNU Scientific Library to generate the Normal distribution and transform it into Con-Nor distribution.

The experiments reported in Figure 3.21 show that the accuracy of RAND is already very high when the sample size reaches 1000; that is, the relative skyline probability error as defined in the last subsection is very low.

The experiment reported in Figure 3.22 evaluate the impact of dimensionality on accuracy. The trends for both Anti-Uniform and Con-Nor-PDF are similar to that for discrete case as depicted in Figure 3.20.

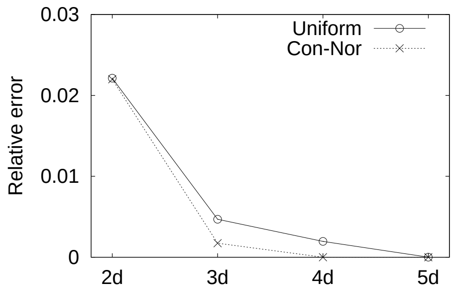


Figure 3.22: Various d

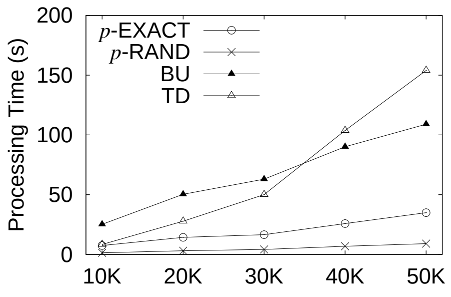


Figure 3.23: Various n

3.6.3 p-skyline computation

We run an experiment to compare the efficiency of p-EXACT and p-RAND with BU and TD algorithms in [PJLY07]. $p = 0.9$ and the 3d Anti-Uniform datasets are employed in the experiment with the number of objects varies from 10K to 50K and $h = 600$. Figure 3.23 shows that p-RAND and p-EXACT significantly outperform BU and TD. Moreover, as the number of objects grows, p-RAND and p-EXACT are quite scalable.

3.6.4 Summary

As a short summary, our experimental results indicate that EXACT is efficient for datasets with a medium number of instances per object. RAND is much more efficient and scalable than EXACT. Meanwhile, it provides high accuracy in both discrete case and continuous case. Moreover, applying these two algorithms to p-skyline problem significantly improves the efficiency of the existing p-skyline techniques.

3.7 Conclusions

In this chapter, we tackle the problem of computing the top-k probabilistic skyline objects on uncertain data. We employ an R-tree index to efficiently conduct the initial computation. Two efficient algorithms are proposed. The exact algorithm aims to precisely rank the top-k skyline objects against skyline probabilities. To deal with the applications where each object has a large set of instances, we develop a randomized algorithm with ϵ-approximation accuracy guarantee, together with a novel, efficient counting algorithm. These two algorithms can be immediately extended to compute p-skyline [PJLY07] to improve the efficiency of the existing techniques and to leading to the first work for computing p-skyline in the continuous case. The extensive experiments demonstrate the efficiency, scalability, and accuracy of our algorithms.

Chapter 4

Probabilistic Skyline Operator over Sliding Windows [1]

Skyline computation over uncertain streaming data has many applications. For instance, in an on-line shopping system products are evaluated in various aspects such as *price*, *condition* (e.g., brand new, excellent, good, average, etc), and *brand*. In addition, each seller is associated with a "trustability" value which is derived from customers' feedback on the seller's product quality, delivery handling, etc. This "trustability" value can also be regarded as occurrence probability of the product since it represents the probability that the product occurs exactly as described in the advertisement in terms of delivery and quality. A customer may want to select a product, say laptops, according to multi-criteria based ranking, such as low price, good condition, and brand preference. For simplicity we assume the customer prefers ThinkPad T61 only and remove the brand dimension from ranking. Table 4.1 lists four qualified results. Both L_1 and L_4 are skyline points,

[1]The techniques presented in this chapter originally appear in the paper "Probabilistic Skyline Operator over Sliding Windows", Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang and Jeffrey Xu Yu, *25th International Conference on Data Engineering (ICDE), 2009*

L_1 is better than (dominates) L_2, and L_4 is better than L_3. Nevertheless, L_1 is posted long time ago; L_4 is better than (dominates) L_3 but the trustability of the seller of L_4 is low.

Table 4.1: Laptop Advertisements.

Product ID	**Time**	**Price**	**Condition**	**Trustability**
L_1	107 days ago	\$ 550	excellent	0.80
L_2	5 days ago	\$ 680	excellent	0.90
L_3	2 days ago	\$ 530	good	1.00
L_4	today	\$ 200	good	0.48

In such applications, customers may want to continuously monitor on-line advertisements by selecting the candidates for the best deal - skyline points. Clearly, we need to "discount" the dominating ability from offers with too low trustability. Moreover, too old offers may not be quite relevant. We model such an on-line selection problem as probabilistic skyline against sliding windows by regarding on-line advertisements as a data stream (see Section 4.1 for details).

Such a data stream may have a very high speed. Consider the stock market application where clients may want to on-line monitor good deals (transactions) for a particular stock. A deal is recorded by two aspects (price, volume) where price is the average price per share in the deal and volume is the number of shares. In such applications, customers may want to know the top deals so far, as one of many kinds of statistic information, before making trade decisions. A deal a is better than another deal b if a involves a higher volume and is cheaper (per share) than those of b, respectively. Nevertheless, recording errors caused by systems or human beings may make unsuccessful deals be recorded successful, and vise versa; consequently each successful deal recorded has a probability to be true. Therefore, a stream of deals may be treated as a stream of uncertain elements and some clients may only want to know "top" deals (skyline) among the most recent N

deals (sliding windows); and we have to take into consideration the uncertainty of each deal. This is another example of probabilistic skyline against sliding windows.

In this chapter we investigate the problem of efficiently processing probabilistic skyline against sliding windows. To the best of our knowledge, there is no similar work existing in the literature in the context of skyline computation over uncertain data steams. In the light of data stream computation, it is highly desirable to develop on-line, efficient, memory based, incremental techniques using small memory. Our contribution may be summarized as follows.

- We characterize the minimum information needed in continuously computing probabilistic skyline against a sliding window.
- We show that the volume of such minimum information is expected to be bounded by logarithmic size in a lower dimensional space regarding a given window size.
- We develop novel, incremental techniques to continuously compute probabilistic skyline over sliding windows.
- We extend our techniques to support multiple pre-given probability thresholds, as well as "top-k" probabilistic skyline.

Besides theoretical guarantee, our extensive experiments demonstrate that the new techniques can support on-line computation against very rapid data streams.

The rest of the chapter is organized as follows. In Section 4.1, we formally define the problem of sliding-window skyline computation on uncertain data streams and present background information. Section 4.2 and Section 4.3 present our theoretic foundation and techniques for processing probability threshold based sliding window queries. Results of comprehensive performance studies are discussed in

Section 4.4. Section 4.5 extends our techniques to top-k skyline, time-based sliding windows, and a data object with multiple instances. Section 4.6 concludes the chapter.

4.1 Background

We use DS to represent a sequence (stream) of data elements in a d-dimensional numeric space such that each element a has a probability $P(a)$ $(0 < P(a) \leq 1)$ to occur where $a.i$ (for $1 \leq i \leq d$) denotes the i-th dimension value. For two elements u and v, u *dominates* v, denoted by $u \prec v$, if $u.i \leq v.i$ for every $1 \leq i \leq d$, and there exists a dimension j with $u.j < v.j$. Given a set of elements, the *skyline* consists of all points which are not dominated by any other element.

4.1.1 Problem Definition

Given a sequence DS of uncertain data elements, a *possible world* W is a subsequence of DS. The probability of W to appear is $P(W) = \Pi_{a \in W} P(a) \times \Pi_{a \notin W}(1 - P(a))$. Let Ω be the set of all possible worlds, then $\sum_{W \in \Omega} P(W) = 1$.

We use $SKY(W)$ to denote the set of elements in W that form the skyline of W. The probability that an element a appears in the skylines of the possible worlds is $P_{sky}(a) = \sum_{a \in SKY(W), W \in \Omega} P(W)$. $P_{sky}(a)$ is called the *skyline probability* of a. The equation (4.1) below can be immediately verified.

$$P_{sky}(a) = P(a) \times \Pi_{a' \in DS, a' \prec a}(1 - P(a')) \tag{4.1}$$

In many applications, a data stream DS is *append-only* [JYC+08, LYWL05, TP06]; that is, there is no deletion of data element involved. In this chapter, we study the skyline computation problem restricted to the append-only data stream model. In a data stream, elements are positioned according to their relative arrival ordering and labelled by integers. Note that the position/label $\kappa(a)$ means that the element a arrives $\kappa(a)$th in the data stream.

Problem Statement. In this chapter, we study the problem of efficiently retrieving skyline elements from the most recent N elements, seen so far, with the skyline probabilities not smaller than a given threshold q ($0 < q \leq 1$); that is, q-skyline. Specifically, we will investigate the problem of efficiently processing such a *continuous* query, as well as *ad-hoc* queries with a probability threshold $q' \geq q$.

4.1.2 Preliminaries

Various Dominating Probabilities. Let DS_N denote the most recent N elements. For each element $a \in DS_N$, we use $P_{new}(a)$ to denote the probability that none of the new arrival elements dominates a; that is,

$$P_{new}(a) = \Pi_{a' \in DS_N, a' \prec a, \kappa(a') > \kappa(a)}(1 - P(a')) \tag{4.2}$$

Note that $\kappa(a') > \kappa(a)$ means that a' arrives after a. We use $P_{old}(a)$ to denote the probability that none of the early arrival elements dominates a; that is,

$$P_{old}(a) = \Pi_{a' \in DS_N, a' \prec a, \kappa(a') < \kappa(a)}(1 - P(a')) \tag{4.3}$$

The following equation (4.4) can be immediately verified.

$$P_{sky}(a) = P(a) \times P_{old}(a) \times P_{new}(a). \tag{4.4}$$

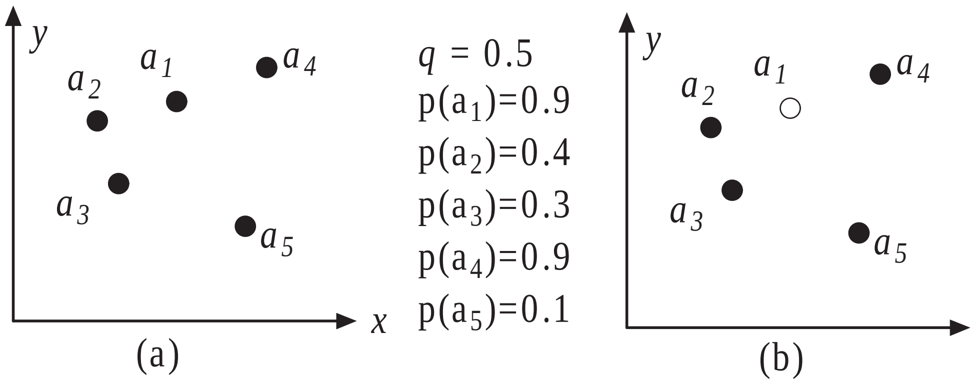


Figure 4.1: A Sequence of Data Elements

Example 4.1. *Regarding the example in Figure 4.1(a) where the occurrence probability of each element is as depicted, assume that* $N = 5$*, and elements arrive according the element subindex order; that is,* a_1 *arrives first,* a_2 *arrives second, ..., and* a_5 *arrives last.* $P_{new}(a_4) = 1 - P(a_5) = 0.9$ *and* $P_{old}(a_4) = (1 - P(a_2))(1 - P(a_3))(1 - P(a_1)) = 0.042$*, and* $P_{sky}(a_4) = P(a_4)P_{new}(a_4)P_{old}(a_4) = 0.034$.

Dominance Relationships. Our techniques will be based on R-trees. Below we define various relationships between each pair of entries E' and E. We use $E.min$ to denote the lower-left corner of the minimum bounding box (MBB) of the elements contained by E, and $E.max$ to denote the upper-right corner of MBB of the elements contained by E. Note that when E degenerates to a single element a, $E.min = E.max = a$.

An entry E *fully dominates* another entry E', denoted by $E \prec E'$, if $E.max \prec E'.min$ or $E.max = E'.min$ with the property that there is no element in E

allocated at $E.max$ or there is no element in E' allocated at $E'.min$. E *partially dominates* E' if $E.min \prec E'.max$ but E does not fully dominates E'; this is denoted by $E \prec_{partial} E'$. Otherwise, E does *not dominate* E', denoted by $E \prec_{not} E'$.

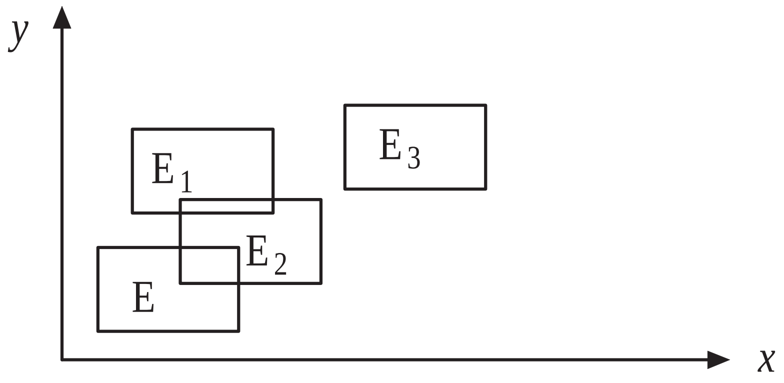


Figure 4.2: Dominance Relationships.

As depicted in Figure 4.2, E fully dominates E_3, and partially dominates E_1 and E_2. Note that E_1 does not dominate E but $E_2 \prec_{partial} E$. Clearly, some elements in E_1 may be dominated by elements in E but elements in E cannot be dominated by any elements in E_1. This can be formally stated below which can be verified immediately according to the definitions.

Theorem 4.1. *Suppose that $E \prec_{partial} E'$. Then some elements in E' might be dominated by elements in E. However, if $E' \prec_{not} E''$. Then elements in E'' cannot be dominated by any element in E'.*

4.2 Framework

Given a probability threshold q and a sliding window with length N, below in Algorithm 4.1 is the framework where a_{old} is the oldest element in current window DS_N and inserting (a_{new}) incrementally computes q-skyline.

Algorithm 4.1 Continuous Probabilistic Skyline Computation over a Sliding Window

```
1: While a new element $a_{new}$ arrives do
2:         if $\kappa(a_{new}) \leq N$ then Inserting ($a_{new}$);
3:         else Expiring ($a_{old}$); Inserting ($a_{new}$);
4: end while
```

Let $S_{N,q}$ denote the set of elements from DS_N with their P_{new} values not smaller than q; that is,

$$S_{N,q} = \{a | a \in DS_N \& P_{new}(a) \geq q\} \tag{4.5}$$

A critical requirement in data stream computation is to have small memory space and fast computation. In our algorithms, instead of conducting the computation against a whole sliding window (N elements), we do the computation restricted to $S_{N,q}$ which will be shown logarithmic in size regarding N on average. Next, we first show the correctness of restricting the computation to $S_{N,q}$.

4.2.1 Using $S_{N,q}$ Only

In this subsection, we will show the following two things: 1) $S_{N,q}$ contains all skyline points with $P_{sky} \geq q$; and 2) computing P_{sky} and P_{new} against $S_{N,q}$ will not lead to *false positive* nor *false negative* to continuously identify $S_{N,q}$ and $SKY_{N,q}$ where $SKY_{N,q}$ is the solution set; that is, for each element a in $SKY_{N,q}$, $P_{sky}(a) \geq q$.

No Missing Elements. The following Lemma is immediate based on (4.4).

Lemma 4.1. *Each q-skyline point a (i.e., $P_{sky}(a) \geq q$) must be in $S_{N,q}$.*

No False Hits to Determine $S_{N,q}$. Suppose that $P_{new}|_{S_{N,q}}(a)$, $P_{old}|_{S_{N,q}}(a)$ and $P_{sky}|_{S_{N,q}}(a)$ denote $P_{new}(a)$, $P_{old}(a)$ and $P_{sky}(a)$ restricted to $S_{N,q}$, respectively.

Example 4.2. *Regarding the example in Figure 4.1, suppose that elements a_1, a_2, a_3, a_4, and a_5 arrive at time 1, 2, 3, 4, and 5, respectively, and $N = 5$, $q = 0.5$. We have that $S_{N,q} = \{a_2, a_3, a_4, a_5\}$ since values of P_{new} for a_2, a_3, and a_5 are the same 1, while $P_{new}(a_4) = 0.9$ as shown in Example 4.1. It can be immediately verified that their P_{new} values restricted to $S_{N,q}$ remain unchanged. Example 4.1 also shows that $P_{old}(a_4) = 0.042$, while $P_{old}(a_4)|_{S_{N,q}} = 0.6 \times 0.7 = 0.42$ since a_1 is not contained in $S_{N,q}$.* $\square$

Next, we show that for each element a in $S_{N,q}$, calculating $P_{new}(a)$ against $S_{N,q}$ is the same as calculating against the whole window DS_N.

Theorem 4.2. *For each element $a \in S_{N,q}$, $P_{new}|_{S_{N,q}}(a) = P_{new}(a)$.*

Theorem 4.2 immediately follows from the following Lemma.

Lemma 4.2. *For each element $a \in S_{N,q}$, if there is an element $a' \in DS_N$ such that $a' \prec a$ and a' is newer than a, then $a' \in S_{N,q}$.*

Proof. Since $a' \prec a$ and a' is newer than a, each element that is newer than a' and dominates a' must dominate a. Consequently, $P_{new}(a) \leq P_{new}(a')$. As $P_{new}(a) \geq q$, $P_{new}(a') \geq q$. Thus, the theorem holds. $\square$

Note that P_{old} values against $S_{N,q}$ are imprecise; nevertheless, below we will show that these will not affect a correct determination of $SKY_{N,q}$.

No False Negative to Determine $SKY_{N,q}$. We show that there is no $a \in SKY_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) < q$.

Theorem 4.3. *For each element $a \in S_{N,q}$, if $P_{old}(a) \times P_{new}(a) \geq q$ then $P_{old}|_{S_{N,q}}(a) = P_{old}(a)$.*

Theorem 4.3 immediately follows the following lemma - Lemma 4.3

Lemma 4.3. *For an element a' such that $a' \prec a$, a' arrives earlier than a, and $P_{old}(a) \times P_{new}(a) \geq q$, then $a' \in S_{N,q}$.*

Proof. Since $a' \prec a$, any element dominating a' must dominate a. Consequently, $P_{new}(a') \geq P_{new}(a) \times P_{old}(a) \geq q$. Thus, $a' \in S_{N,q}$. □

Note that $P_{sky}(a) = P(a)P_{old}(a)P_{new}(a)$ where $P(a) \leq 1$. This, together with Lemma 4.1, Theorems 4.2 and 4.3, immediately implies the following corollary.

Corollary 4.1. *For each element $a \in S_{N,q}$, if $P_{sky} \geq q$ then $P_{sky}(a) = P_{sky}|_{S_{N,q}}(a)$.*

Corollary 4.1 immediately implies there is no false negative; that is, there is no $a \in SKY_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) < q$.

No False Positive to Determine $SKY_{N,q}$. We show that there is no $a \in S_{N,q}$ such that $P_{sky}|_{S_{N,q}}(a) \geq q$ and $P_{sky}(a) < q$.

Theorem 4.4. *For each element $a \in S_{N,q}$, if $P_{old}(a) \times P_{new}(a) < q$, then $P_{old}|_{S_{N,q}}(a) \times P_{new}|_{S_{N,q}}(a) < q$.*

Proof. If every element dominating a is in $S_{N,q}$ then $P_{old}(a)|_{S_{N,q}} \times P_{new}(a)|_{S_{N,q}} = P_{old}(a) \times P_{new}(a) < q$. The theorem holds.

Suppose that at least one element that dominates a is not in $S_{N,q}$. From Lemma 4.2, all such elements must be older than a. Let $Dom(a)$ denote the set of elements that dominate a and are not in $S_{N,q}$. Suppose that a' is the youngest element in $Dom(a)$. It is clear that all elements, which arrive after a' and dominate a', must be contained by $S_{N,q}$ since they dominate a and younger than a'.

Note that $P_{new}(a') < q$. Consequently, $q > P_{new}(a') \geq P_{old}|_{S_{N,q}}(a) \times P_{new}|_{S_{N,q}}(a)$. □

Note that $P_{sky}(a) = P(a)P_{old}(a)P_{new}(a)$ and $P(a) \leq 1$. These, together with Theorems 4.2, 4.3, and 4.4, immediately imply the following corollary.

Corollary 4.2. *For each element $a \in S_{N,q}$, if $P_{sky}|_{S_{N,q}}(a) < q$, then $P_{sky}(a) < q$.*

Therefore, in our techniques we only need to maintain $S_{N,q}$, calculate all probabilities against $S_{N,q}$, and select elements a with $P_{sky}|_{S_{N,q}}(a) \geq q$. For notation simplification, in the remaining of the chapter, $P_{sky}|_{S_{N,q}}$, $P_{old}|_{S_{N,q}}$, and $P_{new}|_{S_{N,q}}$ are **abbreviated to** P_{sky}, P_{old}, P_{new}, respectively if there is no ambiguity.

4.2.2 Estimating sizes of $S_{N,q}$ and $SKY_{N,q}$

Minimality. It can be immediately verified that in order to avoid getting a wrong solution, $S_{N,q}$ is the **minimum** information to be maintained.

Theorem 4.5. *Each element a in the current $S_{N,q}$ with $P(a) \times P_{new}(a) < q$ will never become a q-skyline point; however, there is a data stream such that removing a away will lead to false positive. Moreover, an $a \in S_{N,q}$ with $P(a) \times P_{new}(a) \geq q$ and $P_{sky} < q$ may become a skyline point if old elements dominating e expire and newly arriving elements do not dominate e.*

Theorem 4.5 is quite intuitive and we omit the proof. Below we give an example.

Example 4.3. *Regarding the example in Figure 4.1 (a), assume that $N = 4$. Considering the first window, there are 4 elements a_1, a_2, a_3, and a_4. $S_{N,q} = \{a_2, a_3, a_4\}$ since $P_{new}(a_1) = 0.6 \times 0.7 < 0.5$, while P_{new} values for a_2, a_3, a_4 are all 1. Note that $P_{sky}|_{S_{N,q}}(a_4) = 0.378$; consequently, a_4 is not a q-skyline point based on the current window.*

Regarding the second window when a_1 expires and a_5 arrives. $S_{N,q} = \{a_2, a_3, a_4, a_5\}$ where $P_{new}(a_4) = 0.9$. Other P_{new} values are 1, $P_{sky}(a_3) = P(a_3) =$

$0.3 < 0.5$, and $P_{sky}(a_4) = 0.34 < 0.5$. If we do not record a_3 and a_4 in $S_{N,q}$, then $P_{sky}(a_4)$ will be calculated as $(1 - P(a_5))P(a_4) > 0.5$ leading to the false result, because $P_{sky}(a_4)$ should be $(1 - P(a_2))(1 - P(a_3))(1 - P(a_5))P(a_4) < 0.5$.

Assume that a_1 and a_2 expire, a_5 is as illustrated, and a_6 does not dominate a_4. Regarding the window containing a_3, a_4, a_5, and a_6, $P_{sky}(a_4) = 0.9 \times (1 - 0.3) \times (1 - 0.1) > 0.5$; thus, a_4 is a skyline point. □

Estimating Sizes. Next we show that the expected sizes of $S_{N,q}$ and $SKY_{N,q}$ are bounded by a logarithmic number regarding N.

Suppose that $\chi_{q,i}$ is a random variable such that it takes value 1 if the ith arrival element is a q-skyline point; and $\chi_{q,i}$ takes 0 otherwise. Clearly, the expected size $E(SKY_{N,q})$ of $SKY_{N,q}$ is as follows.

$$E(SKY_{N,q}) = E(\sum_{i=1}^{N} \chi_{q,i}) = \sum_{i=1}^{N} P(\chi_{q,i} = 1) \tag{4.6}$$

Let $I_N = \{j | 1 \le j \le N\}$. Given a set of N probability values $\{P_j | 1 \le j \le N \ \&\ 0 < P_j \le 1\}$, let $P(\neg w) \triangleq \prod_{j \in W}(1 - P_j)$ where W is a subset of I_N. Let $P(W \prec i)$ denote the probability that the ith element is dominated and only dominated by the elements in $\{a_j | j \in W\}$.

Theorem 4.6. *Let DS_N be a sequence of N data elements with probabilities P_1, P_2, ... , P_N. Then,*

$$E(SKY_{N,q}) = \sum_{\forall W, i \notin W, P_i \times P(\neg W) \ge q} P(W \prec i) \times P_i \times P(\neg W) \tag{4.7}$$

Below we show that (4.7) is bounded by a logarithmic size. Given a P_i, let $q_{k,i} \triangleq \max\{P_i \times P(\neg W) | \, |W| = k\}$. Removing the probability value from each data element in DS_N to make DS_N be a sequence DS_N^c of N certain data elements. Let $P(DOM_i^k)$ denote the probability that there are exactly k elements in DS_N^c dominating an element i. The following lemma immediately follows from (4.6). Clearly, $q_{k,i}$ is monotonically decreasing regarding k; that is, $q_{k',i} >= q_{k,i}$ if $k' < k$. Let k_i denote the largest integer such that $q_{k,i} \geq q$ for a given q.

Lemma 4.4. $E(SKY_{N,q}) \leq \sum_{i=1}^{N} \sum_{j=0}^{k_i} P(DOM_i^j) \times q_{j,i}$.

Let $P(DOMT_i^k)$ denote the probability that there are at most k elements dominating the element i. Clearly, $P(DOM_i^k) = P(DOMT_i^k) - P(DOMT_i^{k-1})$.

Corollary 4.3.

$$\begin{aligned} E(SKY_{N,q}) &\leq \sum_{i=1}^{N}(\sum_{j=0}^{k_i-1} P(DOMT_i^j) \times (q_{j,i} - q_{(j+1),i}) \\ &+ P(DOMT_i^{k_i}) q_{k_i,i}). \end{aligned} \tag{4.8}$$

Let $H_{1,l} = \sum_{i=1}^{l} \frac{1}{i}$. The d-th order harmonic mean (for integers $d \geq 1$ and $l \geq 1$) is $H_{d,l} = \sum_{i=1}^{l} \frac{H_{d-1,i}}{i}$. The theorem below presents the value of $P(DOMT_i^k)$.

Theorem 4.7. *For a sequence DS_N^c of N certain data points in a d-dimensional space, suppose that the value distribution of each element on any dimension is the same and independent. Moreover, we assume the values of the data elements in each dimension are distinct. Then, $P(DOMT_i^k) \leq \frac{k+1}{N} \times (1 + H_{d-1,N} - H_{d-1,k+1})$ when $d \geq 2$ and $P(DOMT_i^k) = (k+1)/N$ when $d = 1$.*

Proof. Without lose of generality, we assume that the data elements in DS_N^c are sorted on the first dimension. Since the value distribution of each element on any dimension is the same and independent, an element has the equal probability to take jth position on the first dimension among total N positions; that is $\frac{1}{N}$ probability to take jth position ($1 \leq j \leq N$) on the first dimension. *Note that when a_i takes jth position. any element takes j'th position cannot dominate a_i if $j' > j$.*

When $d = 1$, element a_i must take the first $(k+1)$ positions to ensure there are at most k other elements dominating a_i. Consequently, $P(DOMT_i^k) = (k+1)/N$.

We use mathematic induction to prove the theorem for $d \geq 2$. For $d = 2$, clearly when a_i takes the first $(k+1)$ positions, there are at most $(k+1)$ other elements dominating a_i. When a_i takes a jth position for $j > k+1$, the conditional probability that there must be at most k elements dominating a_i is $\frac{k+1}{j}$ since for each permutation with a_i at jth position on the first dimension, the the value of a_i on the second dimension must take one of the $(k+1)$ smallest value among the j elements with the j smallest values on the first dimension. Thus, we have:

$$\begin{aligned} P(DOMT_i^k) &= \frac{(k+1)}{N} + \frac{1}{N}(\sum_{j=k+2}^{N} \frac{k+1}{j}) \\ &= \frac{k+1}{N} \times (1 + H_{1,N} - H_{1,k+1}) \end{aligned} \tag{4.9}$$

Assume that the theorem holds for $d = l$. For $d = l + 1$, it still holds that when a_i's value on the first dimension is allocated at the first $(k+1)$ positions, then there must be at most k other elements dominating a_i. When a_i takes a jth position for $j > k+1$, the conditional probability that there are at most k elements dominating a_i is $P(DOM_i^k)_{j,l}$ regarding a l-dimensional space and j elements for each permutation with a_i at jth position on the first dimension. Based on our assumption, $P(DOM_i^k)|_{j,l} \leq \frac{k+1}{j} \times (1 + H_{l-1,j} - H_{l-1,k+1})$; consequently, the $P(DOM_i^k)$ regarding the $(l+1)$-dimensional space and N data elements is:

$$P(DOMT_i^k) \leq \frac{k+1}{N} + \frac{1}{N}\sum_{j=k+2}^{N} \frac{k+1}{j} \times (1 + H_{l-1,j} - H_{l-1,k+1})$$

Since $1 \leq H_{l-1,k+1}$, we have:

$$\begin{aligned} P(DOMT_i^k) &\leq \frac{k+1}{N} + \frac{1}{N}\sum_{j=k+2}^{N} \frac{k+1}{j} \times (H_{l-1,j}) \\ &= \frac{k+1}{N}(1 + H_{l,N} - H_{l,k+1}) \end{aligned}$$

□

It can be immediately verified that $H_{d,N} = O(\ln^d N)$; consequently $P(DOMT_i^k) = O(k \ln^{d-1} N)$. This together with Theorem 4.7 and Corollary 4.3 immediately implies that the expected size of $SKY_{N,q}$ in a d-dimensional space is poly-logarithmic regarding N with order $(d-1)$.

Size of $S_{N,q}$. Elements in the candidate set can be regarded as skyline points in a $(d+1)$-space by including the time as an additional dimension since P_{new} can be regarded as the non-dominance probability in such a $(d+1)$-space. We have the following theorem.

Theorem 4.8. *In a d-dimensional space, suppose that the distribution on each dimension, including arriving order are independent. On each dimension, the values of the data items are distinct. Let $P(skyt_i^j)$ denote the probability that there are at most j elements in DS_N^c (remove element probabilities from DS_N) dominating the ith element. Let* $p_{k,i} \triangleq \max\{P(\neg W) | \, |W| = k\}$

$$\begin{aligned} E(SKY_{N,q}) &\leq \sum_{i=1}^{N}\sum_{j=0}^{k_i-1} P(skyt_i^j) \times (p_{j,i} - p_{(j+1),i}) \\ &+ P(skyt_i^{k_i}) p_{k_i,i}. \end{aligned} \tag{4.10}$$

Note that $P(skyt_i^{k_i})$ can be estimated in the same way as that in Theorem 4.7 by replacing d by $d+1$. Therefore, the expected size of $S_{N,q}$ is poly-logarithmic regarding N with the order of d.

4.3 Algorithms

A trivial execution of Algorithm 4.1 is to visit each element in $S_{N,q}$ to update skyline probability when an element inserts or deletes; then choose elements a from $S_{N,q}$ with $P_{sky}(a) \geq q$. Note a new data element may cause several elements to be deleted from $S_{N,q}$, nevertheless, the amortized time complexity is $O(|S_{N,q}|)$ per element which is poly-logarithmic regarding N with the order of d (Section 4.2.2).

In this section, we present novel techniques to efficiently execute Algorithm 4.1 based on aggregate-R trees with the aim to visit as few elements as possible. We continuously, incrementally maintain $SKY_{N,q}$ and $S_{N,q}$.

The rest of the section is organized as follows. We first present data structures to be used. Then we present our efficient techniques to deal with the arrival of a new element for a given probability threshold. This is followed by our techniques to deal with the expiration of an old element for a given probability threshold. Then, we extend our techniques to deal with applications where multiple probability thresholds are given. Finally, correctness and complexity of our techniques are shown.

4.3.1 Aggregate R-trees

Since $SKY_{N,q} \subseteq S_{N,q}$, we continuously maintain $SKY_{N,q}$ and $(S_{N,q} - SKY_{N,q})$ to avoid store a data element twice.

In-memory R-trees R_1 and R_2 on $SKY_{N,q}$ and $(S_{N,q} - SKY_{N,q})$, respectively will be used and continuously maintained. We aim to conduct an efficient computation. Thus, we develop in-memory aggregate R-trees based on the following observation.

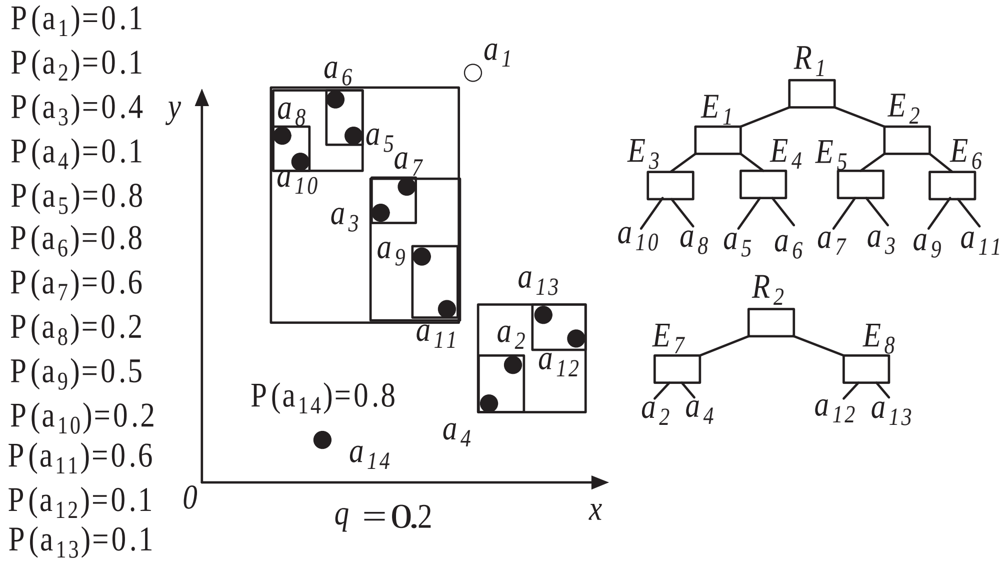


Figure 4.3: Aggregate R-trees

Observation. Regarding the example in Figure 4.3, assume that $N = 13$, $q = 0.2$, the occurrence probabilities are as depicted, and $DS_N = \{a_i | 1 \leq i \leq 13\}$. Suppose that elements arrive according to the increasing order of elements sub-indexes. It can be immediately verified that $P_{new}(a_1) < 0.2$, $S_{N,q}$ contains a_i for $2 \leq i \leq 13$, and $SKY_{N,q}$ contains only the elements in R_1. Two R-trees are built: 1) R_1 is built against the elements in $SKY_{N,q}$; and 2) R_2 is built against the elements in $(S_{N,q} - SKY_{N,q})$.

When a new element a_{14} arrives and a_1 expires. We need to find out the elements which are dominated by a_{14} and then to determine the elements which need to be removed from $S_{N,q}$ and $SKY_{N,q}$. In fact a_{14} dominates entries E_4, E_2, and $R2.root$ (root entry of R_2). If we keep the maximum and minimum values of P_{new} for the elements contained by those entries, respectively, we have a chance not to visit the elements of those entries. Specifically, at an entry if the maximum values of P_{new} multiplied by $(1-P(a_{14}))$ smaller than q, the entry (i.e. all elements contained) will be removed from $S_{N,q}$. On the other hand if the minimum value of P_{new} multiplied by $(1 - P(a_{14}))$ is not smaller than q, then the entry (i.e. all elements contained) remains in $S_{N,q}$. Similarly, at each entry we keep the minimum and maximum values of P_{sky} for the elements contained to possibly terminate the determination of whether elements contained are in $SKY_{N,q}$.

Moreover, in this example elements contained by E_2 is in $S_{N,q}$, we can update their P_{new} values globally by keeping a global value $P_{new}^{global} = P_{new}^{global} \times (1 - P(a_{14}))$ at E_2 to avoid individually update all elements contained in E_2.

Furthermore, in this example a_2 will be removed from $S_{N,q}$ once a_{14} arrives. To avoid update each element contained by E_8 individually due to the removal of a_2, we can keep a global value $P_{old}^{global} = P_{old}^{global} \times (1-P(a_2))$ at E_2 so that we know that the P_{old} values for elements in E_2 will be updated by multiplying $\frac{1}{P_{old}^{global}}$. From time to time, we may remove an entry E from $S_{N,q}$ and E fully dominates another entry E' which stays in $S_{N,q}$. If we keep the no-occurrence probability of the elements in E - $P_{noc} = \Pi_{a \in E}(1 - P(a))$, then we can update P_{old}^{global} at E' by multiplying P_{noc}.

Aggregate Information. Motivated by the observation above, we maintain R_1 and R_2 as aggregate R-trees to keep the above information at each entry. We summarize it below.

- At each entry E, the following information will be stored. $P_{new}^{global}(E)$ stores the captured multiplication of non-occurrence probabilities of the elements which dominate all elements rooted at E. $P_{old}^{global}(E)$ stores the multiplication of non-occurrence probability of the elements that expired and dominate the elements rooted at E.

- At each entry E, we use $P_{noc}(E)$ to store $\prod_{e \in E}(1 - P(e))$.

- At each entry E, $P_{sky,min}(E)$ and $P_{sky,max}(E)$ store the minimum skyline probability and maximum skyline probability of the elements rooted at E without including P_{old}^{global} and P_{new}^{global} at E. $P_{new,min}(E)$ and $P_{new,max}(E)$ store the minimum and maximum P_{new} values of the elements rooted at E without including P_{new}^{global} at E.

Example 4.4. *Continue the example in Figure 4.3 against the first* 13 *elements.*

P_{old}^{global} and P_{new}^{global} at each internal entry are initialized to 1. *When a_{10} arrives, we update $P_{new}^{global}(E_4)$ from* 1 *to* $(1 - P(a_{10})) = 0.8$ *since a_{10} dominates the MBB of E_4, while other P_{new}^{global} values remain* 1.

Here, $P_{noc}(E_3) = (1 - P(a_{10}))(1 - P(a_8)) = 0.64$. Similarly, we can calculate values of P_{noc} at entries E_4, E_5, and E_6. Then, $P_{noc}(E_1) = P_{noc}(E_3) \times P_{noc}(E_4)$ and $P_{noc}(E_2) = P_{noc}(E_5) \times P_{noc}(E_6)$. The multiplication of $P_{noc}(E_1)$ and $P_{noc}(E_2)$ gives P_{noc} at the root. Similarly, P_{noc} values at each internal entry in R_2 can be calculated.

The information that a_{10} dominates both a_5 and a_6 has not been pushed down to leaf-level and is only captured at the entry E_4; consequently the captured skyline probabilities for a_6 and a_5 are $P(a_6) \times (1 - P(a_8))$ (0.64) and $P(a_5)$ (0.8). Therefore, at E_4, $P_{sky,max} = 0.8$ and $P_{sky,min} = 0.64$; $P_{new,max} = 1$ and $P_{new,min} = (1 - P(a_8))$ (0.8). These multiplied by P_{new}^{global} give the exact values of $P_{sky,max}$, $P_{sky,min}$, $P_{new,max}$, and $P_{new,min}$ at E_4, respectively. At other entries, $P_{sky,max}$, $P_{sky,min}$, $P_{new,max}$ and $P_{new,min}$ take exact values.

Once a_2 removes, at E_8, P_{old}^{global} is updated from 1 *to* $(1 - P(a_2)) = 0.9$. □

Removing an Entry. When an entry E removes from R_1 or R_2, we first push down the aggregate information along the path from the root to E and update the siblings' aggregate information for each entry on the path. For example, when remove E_3, we first recalculate the max and min probabilities at the root by CalProb ($R.root$), Algorithm 4.2. Then we push-down P_{new} and P_{old} to E_1 and E_2, respectively by UpdateOldNew ($R_1.root$, E_1) and UpdateOldNew ($R_1.root$, E_2) (Algorithm 4.3). Then we reset P_{old}^{global} and P_{new}^{old} at $R.root$ by 1. We perform the same operations from E_1 to E_3 and E_4.

Algorithm 4.2 CalProb (E)

```
1: if {$P_{old}^{global}(E) < 1$} then
2:     update $P_{sky,min}(E)$, $P_{sky,max}(E)$ by multiplying $\frac{1}{P_{old}^{global}}$;
3: if {$P_{new}^{global}(E) < 1$} then
4:     update $P_{sky,min}(E)$, $P_{sky,max}(E)$, $P_{new,min}(E)$, $P_{new,max}(E)$ by multiplying $P_{new}^{global}$;
```

Algorithm 4.3 UpdateOldNew (E, E')

```
1: if {$P_{old}^{global}(E) < 1$} then
2:     $P_{old}^{global}(E') := P_{old}^{global}(E') \times P_{old}^{global}(E)$;
3: if {$P_{new}^{global}(E) < 1$} then
4:     $P_{new}^{global}(E') := P_{new}^{global}(E') \times P_{new}^{global}(E)$;
```

After E removes from R, we recalculate min and max probabilities, as well as P_{noc} along the path in a bottom-up fashion from E.

Inserting an Entry. In our algorithm, we may need to remove an entry from R_1 and insert it to R_2, and vice versa. When an entry E inserts into R_1 (or R_2), we find an appropriate level to insert E; that is, the level with the length to the leaf to be the same as the depth of E. We also first push down the aggregate information, in the same way as a deletion, to the level. After inserting E, we also recalculate the same aggregate information in the same way as that in a deletion.

Re-balancing. When a re-balancing of R_1 or R_2 as an R-tree is called, we treat it as a deletion followed by an insertion.

4.3.2 Inserting a New Element

As depicted in the last subsection, once a new element a_{new} arrives, we need to conduct the following tasks: 1) update P_{new} values of the elements dominated by a_{new} by multiplying $(1 - P(a_{new}))$, 2) remove the elements a with updated $P_{new}(a) < q$ from R_1 and R_2, 3) update P_{sky} (via P_{old} and P_{new}) values for the elements dominated by some of those removed elements, 4) move elements a in R_1 with $P_{sky}(a) < q$ to R_2, and 5) calculate $P_{sky}(a_{new})$ and insert it to R_1 or R_2 accordingly since $P_{new}(a_{new}) = 1$.

According to Lemma 4.2, if a remaining element a in $S_{N,q}$ is dominated by a removed element a', then a' must be older than a; consequently in the task 3) above, we only need to update P_{old} values. Moreover, by dominance transitivity all the tasks 1) - 4) only need to be conducted against the elements dominated by a_{new}. Clearly, the task 5) is conducted against entries/elements which dominate a_{new}. Therefore, it is critical to identify entries/elements in R_1 and R_2 which are fully dominated by a_{new}, as well as the entries/elements which dominate a_{new}. Algorithm 4.4 is an outline of our techniques.

In Algorithm 4.4, we use $C1$ to store the entries partially dominate a_{new}, $C2$ to store the entries partially dominated by a_{new}, and $C12$ to store the entries which are partially dominated by a_{new} and partially dominate a_{new}. Then, we use Probe ($C1$, $P_{sky}(a_{new})$) and Probe ($C12$, R, $P_{sky}(a_{new})$) to traverse the two aggregate R-trees to get all entries/elements dominating a_{new}. We also use Probe ($C2$, R) and Probe ($C12$, R, $P_{sky}(a_{new})$) to traverse the two aggregate R-trees to get all entries/elements fully dominated by a_{new} and put in R. Finally, UpdateProb

Algorithm 4.4 Inserting (a_{new})

Input: N: window size; q: skyline probability threshold. a_{new}: data element. R_1 and R_2: two aggregate trees on $SKY_{N,q}$ and ($S_{N,q} - SKY_{N,q}$) respectively.

Output: Updated R_1 and R_2

```
1: $P_{sky}(a_{new}) := P(a_{new})$; $P_{old}(a_{new}) := 1$; $P_{new}(a_{new}) := 1$;
2: for each $E \in \{R_1.root, R_2.root\}$ do
3:         if $\{E \prec a_{new}\}$ then
4:                 $P_{sky}(a_{new}) := P_{sky}(a_{new}) \times P_{noc}(E)$;
5:                 $P_{old}(a_{new}) := P_{old}(a_{new}) \times P_{noc}(E)$;
6:         else
7:                 if $a_{new} \prec E$ then add $E$ to $R$;
8:                 if $E \prec_{partial} a_{new}$ & $a_{new} \prec_{not} E$ then add $E$ to $C1$;
9:                 if $E \prec_{partial} a_{new}$ & $a_{new} \prec_{partial} E$ then add $E$ to $C12$;
10:                if $a_{new} \prec_{partial} E$ & $E \prec_{not} a_{new}$ then add $E$ to $C2$;
11: end for each
12: if $C1 \neq \emptyset$ then Probe ($C1$, $P_{sky}(a_{new})$);
13: if $C2 \neq \emptyset$ then Probe ($C2$, $R$);
14: if $C12 \neq \emptyset$ then Probe ($C12$, $R$, $P_{sky}(a_{new})$);
15: if $R \neq \emptyset$ then UpdateProb ($R$);
16: if $P_{sky}(a_{new}) \geq q$ then Add $a_{new}$ to $R_1$; else add $a_{new}$ to $R_2$;
```

(R) conducts tasks 2)-4) and the task 5) is conducted in line 16 by the inserting operation to an aggregate R-tree (R_1 or R_2) as described in Section 4.3.1. Next, we provide details for the procedures Prob () and UpdateProb().

Probe ($C1$, $P_{sky}(a_{new})$) (Algorithm 4.5). According to Theorem 4.1, entries in C_1 cannot contain any element which is dominated by a_{new}. Probe ($C1$, $P_{sky}(a_{new})$) is to iteratively traverse the aggregate R-trees to get entries which dominate a_{new} and then update P_{sky} and P_{old} of a_{new}. In Algorithm 4.5, we use Dequeue () combining with UpdateOldNew () (Algorithm 4.3) to push down the aggregate information. Algorithm 4.6 gives details of Dequeue ().

Probe ($C2$, R). Note that entries in $C2$ do not contain any elements that dominate a_{new} according to Theorem 4.1. Similarly, Probe ($C2$, R) is to iteratively traverse to get all entries/elements which are dominated by a_{new} and then place them in R. As a by-product, we push down the aggregate information and update P^{global}_{new} values of those entries/elements in R. The details are presented in Algorithm 4.7.

Probe ($C12$, R, $P_{sky}(a_{new})$). Entries in $C12$ partially dominate a_{new} and are

Algorithm 4.5 Probe ($C1$, P_{sky})

```
1: While $C1 \neq \emptyset$ do
2:     $E$ := Dequeue ($C1$);
3:     for each Children $E'$ of $E$ do
4:         UpdateOldNew ($E$, $E'$);
5:         if $E' \prec a_{new}$ then
6:             $P_{sky}(a_{new}) := P_{sky}(a_{new}) \times P_{noc}(E')$;
7:             $P_{old}(a_{new}) := P_{old}(a_{new}) \times P_{noc}(E')$;
8:         else
9:             if $E' \prec_{partial} a_{new}$ then add $E'$ to $C1$;
10:        if $E'$ is the last child of $E$ then
11:            reset $P^{global}_{new}(E)$ and $P^{global}_{old}(E)$ to 1;
12: end while
13: return $P_{sky}(a_{new})$;
```

also partially dominated by a_{new}. Consequently, elements contained by entries in $C12$ might dominate a_{new} or are dominated by a_{new}. Probe ($C12$, R, $P_{sky}(a_{new})$), combing with Algorithms 4.5 and 4.7, is to iteratively traverse the aggregate R-trees to possibly further update $P_{sky}(a_{new})$ and add more to R. We present the details below in Algorithm 4.3.2.

UpdateProb (R). R contains all entries/elements which are fully dominated by a_{new} and obtained by Probe ($C12$, R, $P_{sky}(a_{new})$) and Probe ($C2$, R). Note that in our implementation, we use a link list to point to all these entries/elements in R. UpdateProb (R) is to traverse those entries in R, along the aggregate R-trees to which they belong, to detect and remove entries/elements with the updated P_{new} values smaller than q. Moreover, it also updates the P_{old} values of remaining elements in R which are dominated by some removed elements, as well as detects

Algorithm 4.6 Dequeue ($C1$)

```
1: if $C1 \neq \emptyset$ then
2:         get an $E$ in $C1$;
3:         CalProb ($E$);
4: return $E$;
```

the remaining elements in R with $P_{sky} < q$. Algorithm 4.9 provides details.

Lines 1-11: Iteratively detect the elements/entries to be removed (i.e. with $P_{new} < \theta$) and put them to R_3.

Lines 12: UpdateOld (R_3, R_4) is to update the values of P_{old}^{global} of elements/entries in R_4 dominated by some in R_3 as follows. For each pair $E1 \in R_3$ and $E2 \in R_4$,

> if $E1$ fully dominates $E2$, then update $P_{old}^{global}(E2)$ by multiplying $P_{noc}(E1)$; otherwise, if $E1$ partially dominates $E2$ then put the children of $E1$ to R_3 and the children of $E2$ to R_4 for the next iteration.

In our implementation, we mark entries from R (i.e., R_3 and R_4) within R_1 and R_2. Then, we use the *synchronous traversal* paradigm [HJR97] to traverse R_3 and R_4 by following the R-tree structures of the entries in R_3 and R_4. Here, we create a dummy root for R_3 with all entries in R_3 to be children of the root; similar treatments are done for R_4.

lines 13: We remove entries/elements in R_3 from R_1 and R_2 as what discussed in Section 4.3.1.

lines 14: Place (R_4) is to determine elements/entries in R_4 to be in R_1 or R_2. In fact, we only need to check $R_4 \cap R_1$ according to Corollaries 4.1 and 4.2; it is conducted as follows. For each entry $E \in R_4 \cap R_1$, we use depth-first search to find out all its highest level decedent entries with $P_{sky,min}$ greater than q - Algorithm 4.10. In lines 10-11 of Algorithm 4.10, we first remove E from R_1 in the way as

Algorithm 4.7 Probe ($C2$, R)

```
1: While $C2 \neq \emptyset$ do
2:     $E$ := Dequeue ($C2$);
3:     for each Children $E'$ of $E$ do
4:         UpdateOldNew ($E'$);
5:         if $a_{new} \prec E'$ then
6:             $P_{new}^{global}(E') := (1 - P(a_{new})) \times P_{new}^{global}(E')$;
7:             add $E'$ to $R$;
8:         else
9:             if $a_{new} \prec_{partial} E'$ then add $E'$ to $C2$;
10:        if $E'$ is the last child of $E$ then reset $P_{new}^{global}(E)$ and $P_{old}^{global}(E)$ to 1;
11: end while
12: return $R$
```

described in Section 4.3.1. Then, we insert E into R_2 in the way as described in Section 4.3.1.

4.3.3 Expiration

Once an element a_{old} expires, we first check if it is in $S_{N,q}$. If it is in $S_{N,q}$ then we need to increase the P_{old} values for elements dominated by a_{old}. After that, we need to determine the elements that need to be moved from R_2 to R_1. Algorithm 4.11 below presents details.

In Algorithm 4.11, Move ($R \cap R_2$) is to move the elements in $R \cap R_2$ with updated skyline probability not smaller than q to R_1. It is executed in the same way as Place (R_4) but replace $R_1 \cap R_4$ by $R \cap R_2$ and move from R_2 to R_1 instead of R_1 to R_2.

Algorithm 4.8 Probe ($C12$, R, $P_{sky}(a_{new})$)

```
1: While $C12 \neq \emptyset$ do
2:     $E$ := Dequeue ($C12$);
3:     for each Children $E'$ of $E$ do
4:         UpdateOldNew ($E'$);
5:         if $a_{new} \prec E'$ then
6:             $P^{global}_{new}(E') := (1 - P(a_{new})) \times P^{global}_{new}(E')$; add $E'$ to $R$;
7:         else
8:             if $a_{new} \prec_{partial} E'$ & $E' \prec_{not} a_{new}$ then add $E'$ to $C2$;
9:             if $a_{new} \prec_{not} E'$ & $E' \prec_{partial} a_{new}$ then add $E'$ to $C1$;
10:            if $a_{new} \prec_{partial} E'$ & $E' \prec_{partial} a_{new}$ then add $E'$ to $C12$;
11:            if $E' \prec a_{new}$ then
12:                $P_{sky}(a_{new}) := P_{sky}(a_{new}) \times P_{noc}(E')$;
13:                $P_{old}(a_{new}) := P_{old}(a_{new}) \times P_{noc}(E')$;
14:        if $E'$ is the last child of $E$ then reset $P^{global}_{new}(E)$ and $P^{global}_{old}(E)$ to 1;
15: end while
16: if $C1 \neq \emptyset$ then Probe ($C1$, $P_{sky}$) (Algorithm 4.5);
17: else return $P_{sky}(a_{new})$;
18: if $C2 \neq \emptyset$ then Probe ($C2$, $R$) (Algorithm 4.7);
19: else return $R$;
```

4.3.4 Multiple Confidences

Continuous queries Different users may specify different confidences. Suppose that users specify k confidences q_1, q_2, ..., q_k where $q_i < q_{i-1}$. Our techniques for a single given confidence can be immediately extended to cover multiple confidences as follows.

Instead of maintaining a single solution set R_1 in Algorithm 4.11, we maintain k solution sets R_1, R_2, ..., R_k such that elements in R_i (for $2 \leq i \leq k$) have

Algorithm 4.9 UpdateProb (R)

```
1: While $R \neq \emptyset$ do
2:         $E$ := Dequeue ($R$);
3:         if $P_{new,min}(E) < q \leq P_{new,max}(E)$ then
4:             for each Children $E'$ of $E$ do
5:                 UpdateOldNew ($E'$, $E$);
6:                 add $E'$ to $R$;
7:                 if $E'$ is the last child of $E$ then
8:                     reset $P^{global}_{new}(E)$ and $P^{global}_{old}(E)$ to 1;
9:         else
10:            if $P_{new,min}(E) \geq q$ then
11:                add $E$ to $R_4$;
12:            else add $E$ to $R_3$;
13: end while
14: if $R_3 \neq \emptyset$ and $R_4 \neq \emptyset$ then UpdateOld ($R_3$, $R_4$);
15: if $R_3 \neq \emptyset$ then Remove ($R_3$);
16: if $R_4 \neq \emptyset$ then Place ($R_4$);
```

the skyline probabilities in $[q_i, q_{i-1})$ where $q_0 = 1$ and R_{k+1} keeps the elements in $(S_{N,q_k} - \cup_{i=1}^{k} R_i)$. Those R_i for $i = 1$ to $(k+1)$ are also maintained as aggregate R-trees with the same aggregate information.

All the techniques from Algorithm 4.11 are immediately applicable except that now in Algorithm 4.9, we need to detect where to place some elements in $R \cap R_i$ for $i \leq k$; that is, we need to consider all R_j for $i < j < k+1$. In Algorithm 4.11, now we need to detect where to move some elements in R_{k+1}; that is, we need to consider R_j (for $1 \leq j \leq k$) instead of just R_1 in the case of single confidence.

Ad-hoc Queries. Users may also issue an ad-hoc query, "find the skyline with skyline probability at least q'". Assume that currently we maintain k skylines as

Algorithm 4.10 Place (R_4)

```
1: While $R_1 \cap R_4 \neq \emptyset$ do
2:     $E :=$ Dequeue $(R_1 \cap R_4)$;
3:     if $P_{sky,min}(E) < q \leq P_{sky,max}$ then
4:         for each Children $E'$ of $E$ do
5:             UpdateOldNew $(E')$;
6:             add $E'$ to $R_1 \cap R_4$;
7:             if $E'$ is the last child of $E$ then
8:                 reset $P_{new}^{global}(E)$ and $P_{old}^{global}(E)$ to 1;
9:     else
10:        if $P_{sky,max}(E) < q$ then Move $E$ from $R_1$ to $R_2$;
```

discussed above and $q' \geq q_k$. Then, we first find an R_i such that $q_i \leq q' < q_{i-1}$; clearly elements $\{R_j\colon\ j < i-1\}$ } are contained in the solution. We can apply the search paradigm in Place (R_4) (Algorithm 4.10) to get all elements in R_i with skyline probabilities $\geq q$ but without updating aggregate probabilities information.

4.3.5 Algorithm Analysis

Correctness. Our sliding window techniques maintain aggregate information against $S_{N,q}$ and then get skyline according to the skyline probabilities restricted to $S_{N,q}$, Theorems, Lemmas and Corollaries in Section 4.2.1 ensure that our algorithms are correct.

Space Complexity. Clearly, in our algorithm we use aggregate-R trees to keep each element in $S_{N,q}$ and each element is kept only once. Thus, the space complexity is $O(|S_{N,q}|)$.

Time Complexity. It seems hard to provide a sensible time complexity analysis;

Algorithm 4.11 Expiring (a_{old})

```
1: if $a_{old} \in S_{N,q}$ then
2:     Remove ($a_{old}$);
3:     for each $E \in \{R_1.root, R_2.root\}$ do
4:         if $a_{old} \prec E$ then
5:             $P_{old}(E) = P_{old}(E)/(1 - P(a_{old}))$; add $E$ to $R$;
6:         else
7:             if $a_{old} \prec_{partial} E$ then add $E$ to $C$;
8:     while $C \neq \emptyset$ do
9:         $E$ := Dequeue ($C$);
10:        for each Children $E'$ of $E$ do
11:            UpdateOldNew ($E'$);
12:            if $a_{old} \prec E'$ then
13:                $P_{old}(E') := P_{old}(E')/(1 - P(a_{old})$; add $E'$ to $R$;
14:            else
15:                if $a_{old} \prec_{partial} E$ then add $E'$ to $C$;
16:            if $E'$ is the last child of $E$ then reset $P^{global}_{new}(E)$, $P^{global}_{old}(E)$ to 1;
17:    end while
18:    if $R \neq \emptyset$ then Move ($R \cap R_2$);
```

nevertheless, our experiment demonstrates the algorithms in this section is much faster than the trivial algorithm against $S_{N,q}$ as what discussed in the beginning of this section.

4.4 Performance Evaluation

In this section, we only evaluate our techniques since this is the first work studying the problem of probabilistic skyline computation over sliding windows. Specifically, we implement and evaluate the following techniques.

SSKY Techniques presented in Section 4.3 to continuously compute q-skyline (i.e., skyline with the probability not less than a given q) against a sliding window.

MSKY Techniques in Section 4.3.4 to continuously computing multiple q-skylines currently regarding multiple given probability thresholds.

QSKY Techniques in Section 4.3.4 to processing an ad-hoc skyline query with a probability threshold.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4GHz dual CPU and 4G memory under Debian Linux. Our experiments are conducted on both real and synthetic datasets.

Real dataset is extracted from the stock statistics from NYSE (New York Stock Exchange). We choose 2 million stock transaction records of Dell Inc. from *Dec 1st 2000* to *May 22nd 2001.* For each transaction, the average price per volume and total volume are recorded. This 2-dimensional dataset is referred to as *stock* in the following. We randomly assign a probability value to each transaction; that is, probability values follows *uniform* distribution. Elements' arrival order is based on their transaction time.

Synthetic datasets are generated as follows. We first use the methodologies in [BKS01] to generate 2 million data elements with the dimensionality from 2 to 5 and the spatial location of data elements follow two kinds of distributions, *independent* and *anti-correlated.* Then, we use two models *uniform* or *normal* distributions to randomly assign occurrence probability of each element to make them be uncertain. In *uniform* distribution, the occurrences probability of each element takes a random value between 0 and 1, while in the *normal* distribution, the mean value P_μ varies from 0.1 to 0.9 and standard deviation S_d is set 0.3. We assign a random order for elements' arrival in a data stream.

Choosing q. q is the probability threshold in evaluating efficiency of query processing. To evaluate SSKY, we use 0.3 as a default value of q, while to evaluate MSKY with k given probability thresholds q_1, ..., q_k, we let these k values evenly spread $[0.3, 1]$. To evaluate QSKY, we issue 1000 queries across $[q, 1]$ where q is the minimum probability threshold when multiple thresholds are pre-given for multiple continuous skylines. We record average time to process these 1000 queries.

Table 4.2 summarizes parameters and corresponding default values. **In our experiments, all parameters take default values unless otherwise specified.**

Table 4.2: System Parameters

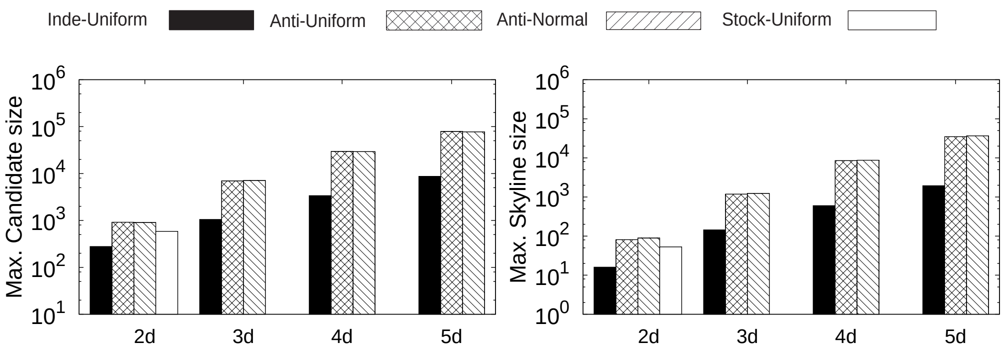
Notation	**Definition (Default Values)**
n	Number of points in the dataset (2M)
N	Sliding Window size (1M)
d	Dimensionality of the of the dataset (3)
D	Dataset (Anti)
D_P	Probabilistic distribution of appearance (*uniform*)
P_μ	expected appearance probability (0.5)
q	probabilistic threshold (0.3)
q'	probabilistic threshold q' ($q \leq q' \leq 1$)

In our experiments, we evaluate the efficiency of our algorithm as well as space usage against dimensionality, size of sliding window, probabilistic threshold, distribution of objects' spatial location and appearance probability distribution.

4.4.1 Evaluate Space Efficiency

We evaluate the space usage in terms of the number of uncertain elements kept in $S_{N,q}$ against different settings. As this number may change as the window slides, we record the maximal value over the whole stream. Meanwhile, we also keep the maximal number of $SKY_{N,q}$.

The first set of experiments is reported in Figure 4.4 where 4 datasets are used: Inde-Uniform (Independent distribution for spatial locations and Uniform distribution for occurrence probability values), Anti-Uniform, Anti-Normal, and Stock-Uniform. We record the maximum sizes of $S_{N,q}$ and $SKY_{N,q}$. It is shown that very small portion of the 2-dimensional dataset needs to be kept. Although this proportion increases with the dimensionality rapidly, our algorithm can still achieve a 89% space saving even in the worst case, 5 dimensional *anti-correlated* data. Size of $SKY_{N,q}$ is much smaller than that of candidates. Since the *anti-correlated* dataset is the most challenging, it will be employed as the default dataset in the following.

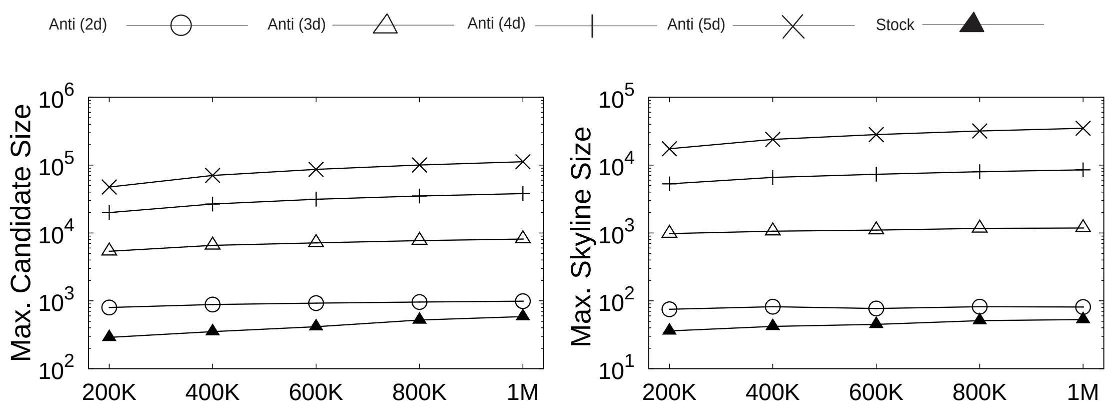


(a) Max. Candidate Size (b) Max. Skyline Size

Figure 4.4: Space Usage vs Diff. Data set

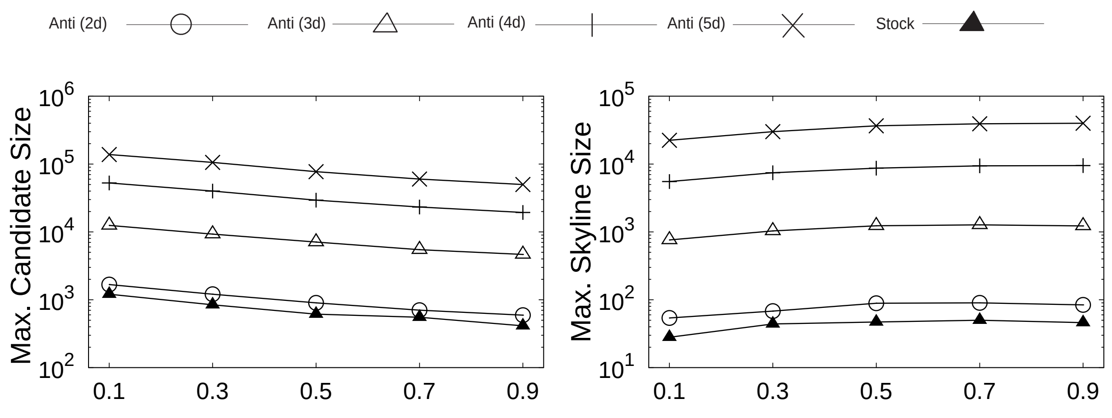
The second set of experiment evaluates the impact of sliding window size N on the space efficiency. As depicted in Figure 4.5, the space usage is sensitive towards the increment of window size.

Figure 4.6 reports the impact of occurrence probability distribution against the space usage and number of skyline points on different datasets. The occurrence probability follows *normal* distribution and the mean of the appearance proba-



(a) Max. Candidate Size(*uniform*)) (b) Max. Skyline Size (*uniform*)

Figure 4.5: Space Usage vs Window Size

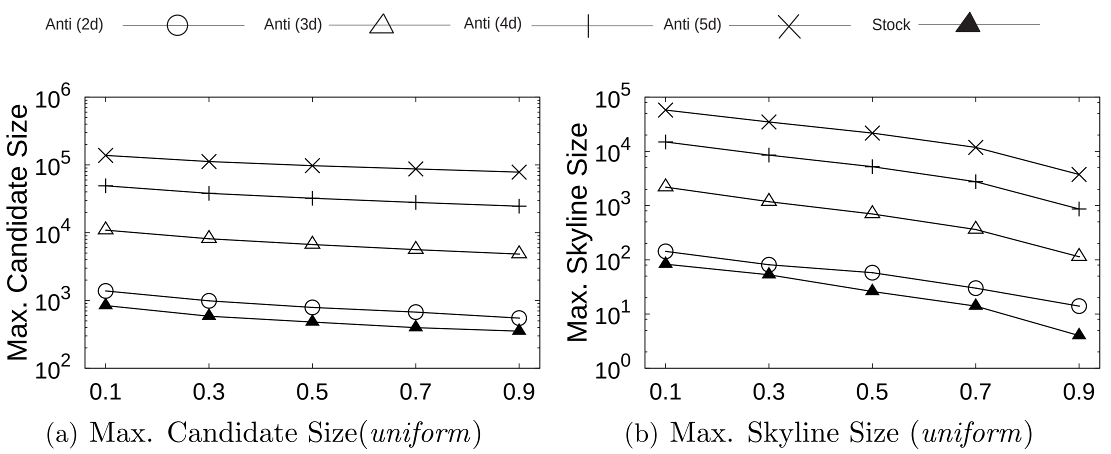


(a) Max. Candidate Size (b) Max. Skyline Size

Figure 4.6: Space Usage vs Appearance Probability

bility P_μ increases from 0.1 to 0.9. It demonstrates that the smaller the average appearance probability of the points, the more points will be kept in $S_{N,q}$. As shown in Figure 4.6(a), the size of the candidate decreases with the increase of average appearance probability. Interestingly, although the candidate size is large with smaller average occurrence probability, the number of probabilistic skyline is small, as illustrated in Figure 4.6(b). This is because the small occurrence probability prevents the uncertain objects from becoming probabilistic skyline.

Figure 4.7 reports the effect of probabilistic threshold q on space efficiency. As expected, both candidate set size and skyline set size drop as q increases.



(a) Max. Candidate Size(*uniform*) (b) Max. Skyline Size (*uniform*)

Figure 4.7: Space Usage vs Probability Threshold

4.4.2 Evaluation Time Efficiency

We evaluate the time efficiency of our continuous query processing techniques, SSKY and MSKY, as well as ad-hoc query processing technique QSKY. We first compare SSKY with the trivial algorithm against $SKY_{N,q}$ as described in the beginning of Section 4.3. We find it is about 20 times slower than SSKY against anti (3d). Thus, we exclude the trivial algorithm from further evaluation.

Since the processing time of one element is too short to capture precisely, we record the average time for each batch of 1K elements to estimate the delay per element.

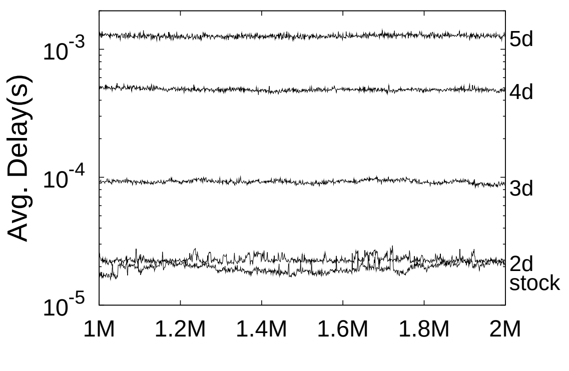


Figure 4.8: Time Efficiency vs n

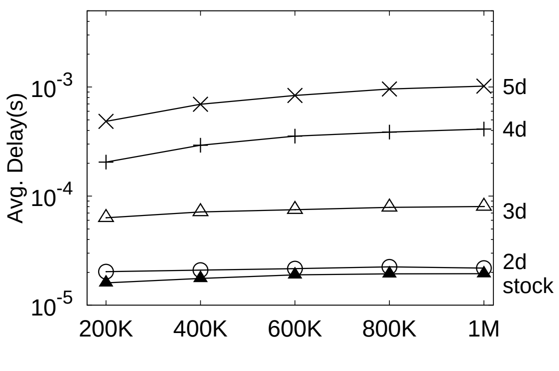


Figure 4.9: Avg. Delay vs W

The first set of experiment is depicted in Figure 4.8. It shows that SSKY is very efficient, especially when the dimensionality is low. For 2 dimensional dataset, SSKY can support a workload where elements arrive at the speed of more than 38K per second even for *stock* and *anti-correlated* dataset. For 5*d* *anti-correlated* data, our algorithm can still support up to 728 elements per second, which is a medium speed for data streams.

Figure 4.9 evaluates the system scalability towards the size of the sliding window. The performance of SSKY is not sensitive to the size of sliding window. This is because the candidate size increases slowly with N, as reported in Figure 4.5.

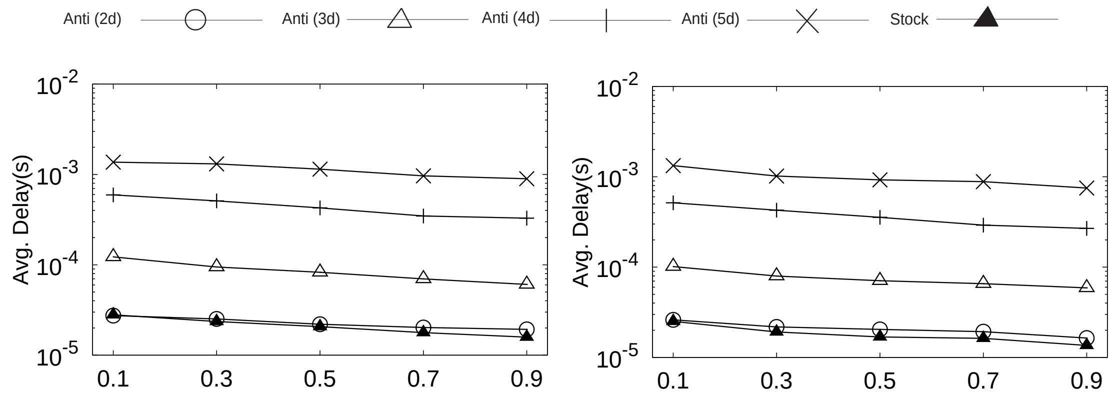


Figure 4.10: Avg. Delay vs P_μ

Figure 4.11: Avg. Delay vs q

Figure 4.10 evaluates the impact of occurrence probability distribution on time efficiency of SSKY where normal distribution is used for probability values. As expected, large P_μ leads to better performance since the candidate size is small when P_μ is large.

Figure 4.11 evaluates the effect of probability threshold q on SSKY. Since both size of candidate set and skyline objects set are small when q is large as depicted in Figure 4.7, SSKY is more efficient when q increases.

The last experiment evaluates the efficiency of our multi probability thresholds based continuous query processing techniques MSKY and ad-hoc query processing techniques. Results are reported in Figures 4.12(a) and 4.12(b), respectively. As

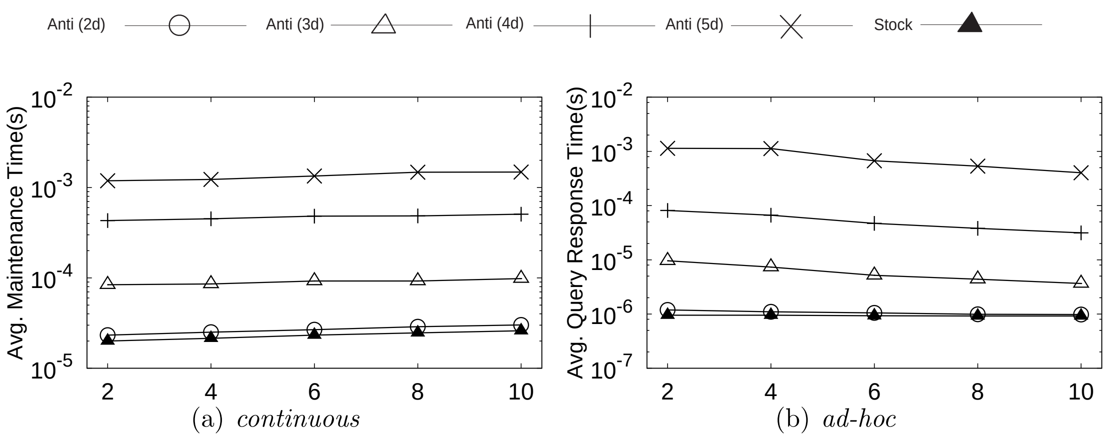


Figure 4.12: Query Cost vs $|Q|$

expected, Figure 4.12(a) shows that cost to process each element by MSKY increases when k increases, while Figure 4.12(b) shows the ad-hoc query processing cost decreases when k increases.

4.4.3 Summary

As a short summary, our performance evaluation indicates that we only need to keep a small portion of stream objects in order to compute the probabilistic skyline over sliding windows. Moreover, our continuous query processing algorithms are very efficient and can support data streams with high speed for 2d and 3d datasets. Even for the most challenging data distribution, *anti-correlated*, we can still support the data stream with medium speed of more than 700 elements per second when dimensionality is 5.

4.5 Applications

The techniques developed in this chapter can be immediately extended to the following applications.

Probabilistic Top-k Skyline Elements. Given an uncertain data stream, a threshold q, and a sliding window size W, find the k skyline points with the highest skyline probabilities (but not smaller than q).

We can apply our algorithms in Section 4.3 to remove points with $P_{new} < q$, update aggregate information at each entry, probabilities (P_{sky}, P_{old}, P_{new}, etc). We do not move any elements in $R_4 \cap R_1$ to R_2. Instead, we treat R_1 and R_2 as two "heap trees". In fact, both R_1 and R_2 maintain two heaps on P_{sky}: 1) min-heap, and 2) max-heap; this is because we keep $P_{sky,min}$ and $P_{sky,max}$ at each entry. We use min-heap on R_1 and max-heap on R_2 to move elements in top-k from R_2 to R_1 and move elements in R_1 but not in top-k to R_2.

Time Stamp based Sliding Windows. In such a model, we expire an old element if it is not within a pre-given most recent time period T. Our techniques can be immediately extended to sliding windows based on the most recent time period T.

Object with Multiple Elements. Suppose that an uncertain stream contains a sequence of objects such that each object consists of a set of instances [PJLY07] or PDF. In fact, our skyline probability model is a special case of the model in [PJLY07]. In our sliding window model, we assume that each object is *atomic*.[2] Then we want to compute objects with skyline probabilities not smaller than q. It can be immediately verified that all our techniques are immediately applicable to discrete cases except we compute skyline probability in a different way; that is, based on the definition in [PJLY07]. For continuous cases, we can use Monte-Carlo sampling method [KW86] to discrete them.

[2]When an object arrives, all its instances arrive; when an object expires, all its instances expire.

4.6 Conclusion

In this chapter, we investigate the problem of efficiently computing skyline against sliding windows over an uncertain data stream. We first model the probability threshold based skyline problem. Then, we present a framework which is based on efficiently maintaining a candidate set. We show that such a candidate set is the minimum information we need to keep. Efficient techniques have been presented to process continuous queries. We extend our techniques to concurrently support processing a set of continuous queries with different thresholds, as well as to process an ad-hoc skyline query. Finally, we show that our techniques can also be extended to support probabilistic top-k skyline against sliding windows over an uncertain data streams. Our extensive experiments demonstrate that our techniques can deal with a high-speed data stream in real time.

Chapter 5

Probabilistic Top-k Dominating Queries [1]

Top-k dominating queries and skyline are shown as useful tools in decision making [BKS01, PTGS03, TEO01, YM07] to rank certain data. A *top-k dominating* query retrieves the k objects with the highest dominating ability, that is, the k objects that dominate the largest number of other objects. It is formally defined as follows [YM07]. Suppose that $\mathcal{X}$ is a set of d-dimensional points. For a point $x \in \mathcal{X}$, the score function is defined as the number of points dominated by x, namely, $score(x) = |\{x' \in \mathcal{X} | x \prec x'\}|$. Here, $x \prec x'$ if the coordinate value of x is not greater than that of x' at each dimension with at least one dimension at which the coordinate value of x is smaller than that of x'. $score(x)$ is a useful ranking function due to the following ordering property [YM07]: $\forall x,\ x' \in \mathcal{X},\ x \prec x' \Rightarrow score(x) > score(x')$. A top-$k$ dominating query retrieves the k points in $\mathcal{X}$ with the highest scores. The *skyline* operator retrieves all objects from $\mathcal{X}$ which are not

[1] The techniques presented in this chapter are originally published in the paper "Threshold-based Probabilistic Top-k Dominating Queries", Wenjie Zhang, Xuemin Lin, Ying Zhang, Jian Pei and Wei Wang, *accepted by VLDB Journal, 2009*

dominated by other objects.

The skyline operator and top-k dominating queries rank objects in different ways: skyline ranks objects in a "defensive" way and outputs the objects which are not worse than any other objects in a given dataset, while a top-k dominating query ranks objects in an "assertive" way and provides the objects that are better than the largest number of other objects. As pointed out in [YM07], the benefit of using top-k dominating queries is to assimilate the advantages of top-k queries and the skyline operator. That is, the result size in a top-k dominating query is strictly controlled by k, while like skyline operators, top-k dominating queries do not require a specific ranking function and are not affected by potentially different scales at different dimensions.

Figure 5.1 shows the average performance of 3 popular NBA players from 3 selected games in their rookie seasons with respect to two statistics aspects, number of assists (AST) and number of points (PTS). To retain the preference of smaller values, we record $(30 - PTS)$ and $(6 - AST)$ in Figure 5.1, while the corresponding three game statistics are depicted in Figure 5.2. According to the aggregate information (average performance), the skyline consists of Shaquille O'neal and Elton Brand and the top-2 dominating query also returns O'neal and Brand in this example. Both dominate Brown but there is no dominating relationship between O'neal and Brand.

Motivating Example. Take NBA players as an example. NBA players may be ranked in various ways. Dominating queries provide an effective way to rank a player according to the number of other players whom this player outperforms. Using aggregates, such as AVERAGE per game, to summarize game statistics and then to count dominating relationships by the top-k dominating computation techniques in [YM07] is an option. While aggregates such as AVERAGE per game

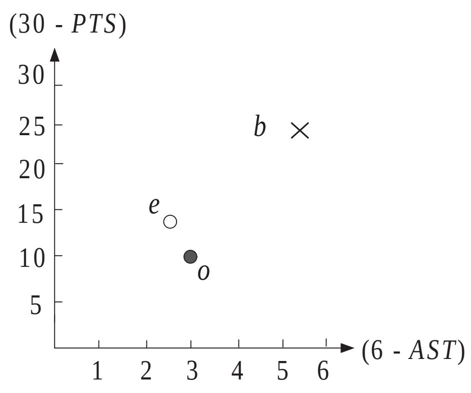


Figure 5.1: Average

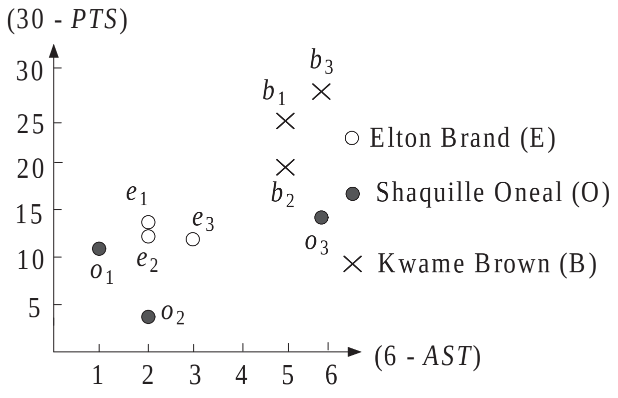


Figure 5.2: NBA Players.

is useful to summarize the statistic information, they do not quite reflect the actual game-by-game performances and may be potentially affected by "outliers".

As depicted in Figure 5.2, O'neal's overall performance is affected by a bad outlier - o_3. Consequently, O'neal ties with Brand if we choose the top-1 dominating player according to the aggregate information in Figure 5.1. However, intuitively O'neal should be the winner based on the game-by-game statistics in Figure 5.2. The examples depicted in Figures 5.1 and 5.2 are quite representative.

We have conducted an evaluation on the fourteen 1st picks from 1991 to 2004 regarding their rookie seasons. To conduct a fair evaluation, we use the first 54 games (i.e. their rookie season games) against 3 kinds of game-by-game statistics, scores, rebounds, and assists since the year 1997 only has 54 games in the regular season. The 2nd column of Table 5.1 illustrates the ranks (bold number) of these players based on the number (the number in the bracket) of players dominated by them, respectively, using the average statistics per player, where Duncan is ranked first, Johnson is ranked 2nd, Webber and Brand are tied at 3rd, and O'neal is ranked 5th. Note that it is commonly believed that O'neal has the best rookie season among those players especially comparing to Brand's rookie season [2]. Thus,

[2]See wikipedia and also `http://armchairgm.wikia.com/Top_No._1_Overall_NBA_Draft_`

the top-k dominating queries against aggregates (average) may not provide right semantics for the applications where each object has multiple "instances" to occur.

Name	Ranks				
	agg	2%	5%	10%	20%
O'neal, S	**5**(4)	**5**(4)	**3**(5)	**5**(4)	**5**(4)
Johnson, L	**2**(6)	**1**(7)	**2**(6)	**1**(8)	**1**(10)
Duncan, T	**1**(7)	**1**(7)	**1**(7)	**2**(7)	**2**(7)
Webber, C	**3**(5)	**3**(5)	**3**(5)	**3**(5)	**3**(5)
Brand, E	**3**(5)	**3**(5)	**3**(5)	**3**(5)	**3**(5)
James, L	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Robinson, G	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Smith, J	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Iverson, A	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Ming, Y	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Howard, D	**6**(2)	**6**(2)	**6**(2)	**6**(2)	**6**(2)
Martin, K	**10**(1)	**10**(1)	**10**(1)	**10**(1)	**10**(1)
Olowokandi, M	**13**(0)	**13**(0)	**13**(0)	**13**(0)	**13**(0)
Brown, K	**13**(0)	**13**(0)	**13**(0)	**13**(0)	**13**(0)

Table 5.1: Ranks of NBA 1st Picks after Removing Outliers

Conducting an aggregate (e.g. average) after removing outliers and then applying the top-k dominating computation technique in [YM07] is a possible paradigm. To verify the affect of such a paradigm, we conduct the experiment on the above rookie data. We first employ one of the most popular clustering algorithms, DBSCAN [EKSX96], to remove 2%, 5%, 10% and 20% of instances as outliers from each player by choosing the distance and density parameters. Then, we calculate the average performance over remaining data for each player and then do the domination counting against the average performance. The result is depicted in Table 5.1 where $x\%$ for $x = 2, 5, 10, 20$ means $x\%$ of outliers have been removed. Table 5.1 shows that removing outliers does not quite affect the above rankings; this is because that there are bad outliers and good outliers. Therefore, the paradigm of removing outliers and then applying the top-k dominating computation may suffer

`Picks`

Name	Ranks									
	agg	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
O'neal, S	**5** (4)	**1** (1)	**1** (2)	**1** (2)	**1** (3)	**1** (4)	**1** (5)	**1** (5)	**1** (6)	**3** (7)
Johnson, L	**2** (6)	**2** (0)	**2** (1)	**1** (2)	**1** (3)	**1** (4)	**1** (5)	**1** (5)	**1** (6)	**1** (8)
Duncan, T	**1** (7)	**2** (0)	**2** (1)	**1** (2)	**1** (3)	**3** (3)	**3** (4)	**1** (5)	**1** (6)	**1** (8)
Webber, C	**3** (5)	**2** (0)	**2** (1)	**4** (1)	**4** (2)	**3** (3)	**3** (4)	**4** (4)	**4** (5)	**3** (7)
Brand, E	**3** (5)	**2** (0)	**5** (0)	**4** (1)	**4** (2)	**5** (2)	**5** (3)	**5** (3)	**5** (4)	**5** (6)
James, L	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**6** (1)	**6** (1)	**6** (2)	**5** (3)	**5** (4)	**6** (5)
Robinson, G	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**6** (1)	**6** (1)	**6** (2)	**7** (2)	**7** (3)	**8** (4)
Smith, J	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**6** (2)	**7** (2)	**7** (3)	**8** (4)
Iverson, A	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**9** (1)	**7** (2)	**7** (3)	**8** (4)
Ming, Y	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**6** (1)	**9** (1)	**7** (2)	**7** (3)	**6** (5)
Howard, D	**6** (2)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**9** (1)	**11** (1)	**11** (2)	**12** (2)
Martin, K	**10** (1)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**9** (1)	**11** (1)	**11** (2)	**11** (3)
Olowokandi, M	**13** (0)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**13** (0)	**13** (0)	**13** (1)	**13** (1)
Brown, K	**13** (0)	**2** (0)	**5** (0)	**6** (0)	**8** (0)	**11** (0)	**13** (0)	**13** (0)	**14** (0)	**14** (0)

Table 5.2: Ranks of NBA 1st Picks

from the following issues.

- The actual distributions of multiple instances are not addressed.
- Since 'bad" and "good" performance outliers have different affects, the contributions of "outliers" are not evaluated.

Probabilistic Dominating Queries. To address the applications where an object has multiple instances (e.g. game statistics of a NBA player), in this chapter we develop a probabilistic model to measure the dominating ability of each object. Unlike conventional dominating queries, from probabilistic point of view each object could dominate any number of objects even with a very small probability (say 0, or close to 0). Therefore, we use the probability q by which an object dominates *at least* l objects to measure the dominating ability; that is, the dominating ability of an object U is measured by two parameters (q, l). Generally, the larger l, the smaller q. Consequently, there are two ways to model a probabilistic dominating query.

1. Given a probability threshold q, for each object U compute the maximum l such that U dominates at least l other objects with probability not smaller than q.

2. Given a threshold l, for each object U compute the maximum q such that U dominates at least l objects with probability not smaller than q.

In the second model, q could be very small. To control the value of q, in this chapter we focus on the first model. Nevertheless our techniques can be immediately applied to the second model; we will discuss this in Section 7. In Table 5.2, we show the ranking results according to the 1st model where we assign each game statistic by the same occurrence probability. The 3rd to 11th columns show the ranks based on different probability thresholds, respectively. For example, in the column headed by the threshold 0.5, O'neal dominates at least 4 (the number in bracket) other players with at least the probability 0.5; thus he is ranked 1st (bold number). Clearly, O'neal is the winner against each of those probability thresholds except when the probability threshold is 0.1 (worse than Duncan and Johnson). The probabilistic rankings catch the common perception better. It is also interesting to note that Duncan dominates 7 players with a probability between 0.1 to 0.2, while according to the average (aggregate) game statistics Duncan actually dominates 7 players. Clearly, the domination counting regarding a small q is largely biased towards to "good" outliers. On the other hand, the phenomenon for a very large q (close to 1) is not very meaningful since q is always 1 if $l = 0$. The most interesting part of q is towards the middle of $(0, 1]$; these values will show the dominating ability among majority instances of objects, respectively. In addition, our probabilistic model provides a tool for us to "drill down" information against different probabilistic threshold values to provide the breakdown information like that in Table 5.2.

Probabilistic Top-k Dominating Queries. In this chapter, we will study the problem of retrieval of k objects with the maximum values of l for a given probability threshold q (i.e., based on the 1st model). We will adopt the assumption that the probability distribution of the object is independent to each other due to the following reasons. Firstly, it is a common model currently adopted in probabilistic query processing. Secondly, handling dependence among a large number of objects is not only complex but also expensive, while applications with the assumption of independent distributions exist. For example, regarding the above example there is no reason to believe a dependence among those 1st picks' performance in their rookie season across different years, given the game rules are the same and the other players in each year have similar talents. Similarly, we could also evaluate the top-k all-round gymnastics players with the same gender by treating each competition record as an instance of a player where each competition record consists of scores for each individual programs. In male competitions, scores from Vault, Floor, Parallel bars, Rings, Pommel horse and Horizontal bar are recorded in each competition, respectively, as a 6-dimensional instance. While the performances of each female player in four programs (Vault, Floor, Uneven bars and Balance beam) are recorded per each competition. Clearly, the performances of the players are independent with each others.

Contributions. As shown above, dominating relationships among uncertain objects are quite complex and probabilistic distribution dependent. Moreover, due to the nature of uncertain data and dominating queries, an expensive computation will be involved in exactly computing "probabilistic" scores of objects; consequently, it is too expensive to compute such scores for all objects.

In this chapter, we investigate the problem of efficiently computing the top-k dominating queries against uncertain objects where object PDFs are not available; that is, we deal with *discrete cases.* To the best of our knowledge, this is the first work addressing the top-k dominating query over uncertain data. Our contributions may be summarized as follows.

- We formally define a top-k dominating query on uncertain data with a given probability threshold imposed to support different confidence requirements.

- An efficient, threshold-based exact algorithm is proposed to take an advantage of the threshold-based paradigm [FLN03]. Based on a novel application of laws of large numbers [Gol01] and mathematic characterizations, a set of novel, effective pruning techniques have been proposed to pursue efficiency.

- We develop an efficient randomized algorithm with an accuracy guarantee. Novel processing techniques and data structures are developed in our randomized techniques.

An extensive experimental study over synthetic and real data shows that our exact algorithm performs well, while our randomized algorithm is not only highly accurate and more efficient than the exact algorithm but also quite scalable against data sizes, object uncertain areas, k values, etc.

Organization. The rest of this chapter is organized as follows. Section 5.1 formally defines probabilistic top-k dominating queries and presents preliminaries. Section 5.2 briefly outlines the framework of our exact and randomized algorithms. In Section 5.3, we present our exact algorithm. Following the framework of exact algorithm, a novel randomized algorithm is presented in Section 5.4. Our experiment results are reported in Section 5.5. This is followed by the discussions regarding the model where a threshold of a domination counting is given and the general cases where probabilistic distributions may be correlated. We conclude this chapter in section 5.6.

5.1 Background Information

We first model the problem and then, present the preliminaries of the chapter. For reference, notations used in this chapter are summarized in Table 6.1.

Notation	**Definition**
$\mathcal{U}$	set of uncertain objects
U, V	uncertain objects
u, v	instances of uncertain objects
E	entry in an aR-tree of objects, instances, and samples
$MBB_U(MBB_E)$	minimum bounding box of $U(E)$
$\mu_U(\mu_E)$	upper-right corner of $MBB_U(MBB_E)$
$\imath_U(\imath_E)$	lower-left corner of $MBB_U(MBB_E)$
$P(\tau)$	probability of τ to occur
q	probability threshold of a query
$pscore(\upsilon)$	probabilistic score of υ ($\upsilon = U$ or u)
$pscore^+$	upper bound of *pscore*
$P_{=l}(U)(P_{=l}(u))$	probabilities to dominate l objects
$P_{\geq l}(U)(P_{\geq l}(u))$	probabilities to dominate $\geq l$ objects
γ_k	minimum *pscore* of the top-k objects
λ_k	minimum *pscore* of the current top-k objects
$P(u \prec V)$	probability of u dominating V
Ω	set of possible worlds
$L_i(U)$	ith level entries in an aR-tree of U
P^{upper}	upper bound of probability P
$PD(U)$ $(FD(U))$	set of objects partially (fully) dominated by U
$PD(E)$ $(FD(E))$	set of entries (objects) partially (fully) dominated by E
$\overrightarrow{\mathcal{U}}$	partially ordered list of uncertain objects

Table 5.3: The Summary of Notations.

5.1.1 Problem Statement.

Our investigation in the chapter will focus on discrete cases. An *uncertain* object U is represented by a set of *instances* such that each instance $u \in U$ is a point in a d-dimensional numeric space $D = \{D_1, ..., D_d\}$ with the probability $P(u)$ to occur where $0 < P(u) \leq 1$ and $\sum_{u \in U} P(u) = 1$.

Given a set of uncertain objects $\mathcal{U} = \{U_1, \cdots, U_n\}$, a *possible world* $W = \{u_1, \cdots, u_n\}$ is a set of n instances - one instance per uncertain object. The probability of W to appear is $P(W) = \prod_{i=1}^{n} P(u_i)$. Let Ω be the set of all possible worlds; that is, $\Omega = U_1 \times U_2 \cdots \times U_n$. Then, $\sum_{W \in \Omega} P(W) = 1$.

$\Omega_{\ell,U}$ denotes the set of possible worlds in each of which the instance $u \in U$ dominates exactly ℓ other instances. Clearly, the probability $P_{=\ell}(U)$ of U dominating exactly ℓ objects is:

$$P_{=\ell}(U) = \sum_{W \in \Omega_{\ell,U}} P(W). \tag{5.1}$$

Example 5.1. *Regarding the example in Figure 5.2, we treat every player as an uncertain object and each game statistic as an instance of the object. Unless specified otherwise, the occurring probability of each instance is* $1/3$. $\Omega_{1,O} = \{\{o_3, e_1, b_3\}, \{o_3, e_2, b_3\}, \{o_3, e_3, b_3\}, \}$. *Dominating probabilities of each player are as follows.*

$P_{=0}(O) = 2/9$, $P_{=1}(O) = 3/27$, $P_{=2}(O) = 2/3$;

$P_{=0}(E) = 0$, $P_{=1}(E) = 2/3$, $P_{=2}(E) = 1/3$;

$P_{=0}(B) = 1$, $P_{=1}(B) = 0$, $P_{=2}(B) = 0$.

As mentioned earlier, unlike dominating queries on certain objects, an uncertain object can dominate any number of objects with some probability. Nevertheless, such dominating probabilities could be very small (even zero); results with a small probability to occur are not very interesting. To resolve this, in our problem definition we enforce a probability threshold, and we model probabilistic dominating queries in an accumulative way; that is, we look for the objects that dominate at least ℓ other objects with at least probability (confidence) q. We assign a probabilistic score, $pscore_q(U)$, to each uncertain object U as follows.

Let $P_{\geq \ell}(U)$ denote the probability of U dominating at least ℓ other objects.

Clearly,

$$P_{\geq \ell}(U) = \sum_{i=\ell}^{n} P_{=i}(U). \tag{5.2}$$

Definition 5.1 ($pscore_q$). *$pscore_q(U)$ is the maximum ℓ such that $P_{\geq \ell}(U) \geq q$.*

Note that for notation simplification, $pscore_q$ is hereafter abbreviated to $pscore$ whenever there is no ambiguity.

Definition 5.2 (PtopkQ). *Given a probability threshold q, an integer k, and a set $\mathcal{U}$ of uncertain objects, PtopkQ retrieves the k objects with the highest pscore values. Ties are broken arbitrarily.*

Example 5.2. *Regarding the example in Figure 5.2 when $q = 2/3$, $pscore_q(O) = 2$, $pscore_q(E) = 1$ and $pscore_q(B) = 0$; that is, O'neal is the top dominating player.*

We will develop efficient exact algorithms as well as efficient and effective randomized algorithms to compute PtopkQ.

5.1.2 Preliminaries

Centroid. The dominating ability of an object is determined by the distribution of its instances and its relationships to the distributions of instances of other objects. The centroid $\omega(U)$ of instances will be used in our algorithms to approximately represent the distribution of instances. Formally, $\omega(U) = \sum_{u \in U} P(u) \times u$.

aR-tree. An aggregate R-tree (aR-tree) [PKZT01] is an extension of R-tree [Gut84] where each entry keeps the number of objects contained. Figure 5.3 illustrates 9 data points indexed by an aR-tree, bounded by 3 MBBs at the leaf level.

Top-k Dominating Query on Certain Data. Given a k and a set of points, the CBT (cost-based traversal) algorithm in [YM07] selects the k points with the

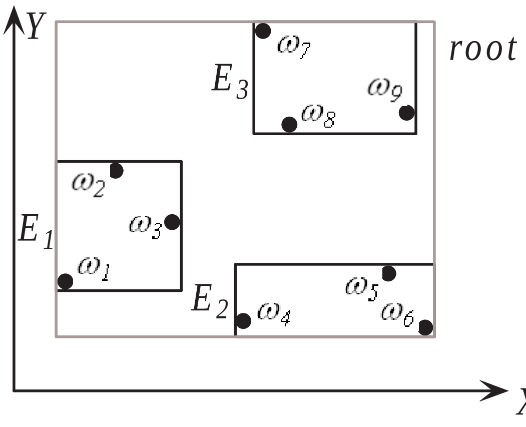


Figure 5.3: Certain Data

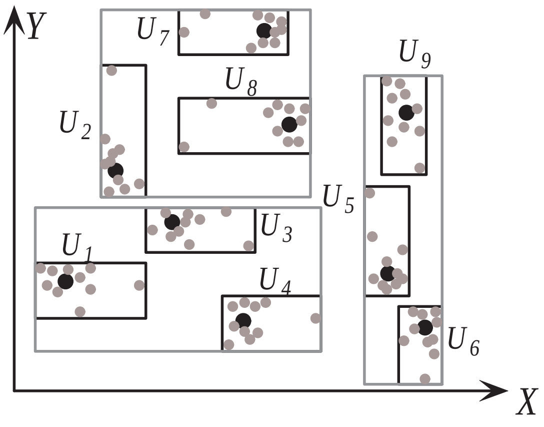


Figure 5.4: Uncertain Data

highest dominating *score* values. Recall that $score(x)$ of a point is the number of other points dominated by x. Below we briefly introduce CBT, to be used as a black-box in the preprocessing of our algorithm in Section 5.3.1.

In CBT, an aR-tree is used. The algorithm CBT traverses the aR-tree level by level to calculate a lower bound $score^-(E)$ and an upper-bound $score^+(E)$ of the number of points dominated by a point in an entry E of aR-tree. An entry E is *pruned* if $score^+(E)$ is not greater than the current kth largest $score^-$ and the points in E are the solution if $score^-(E)$ is not smaller than the current kth largest $score^+$; otherwise E will be drilled down to the lower level; these are conducted by taking the consideration of the number of points in intermediate entries of aR-tree. In our preprocessing, we will make use of the entries that are either pruned or stayed in the job queue when the algorithm CBT terminates. Clearly, these entries are disjoint and cover all points. Note that these entries can be either points or intermediate entries. Below is an example.

Example 5.3. *Regarding the example in Figure 5.3, if k=2, the algorithm terminates with the following entries in the job queue:*

$$\{\omega_1.[6,6], \omega_4.[4,4], \omega_2.[3,3], \omega_3.[3,3], \omega_5.[0,3], \omega_6.[0,0]\},$$

while $E_3.[0,2]$ is pruned. In each entry representation, the left-end in the bracket is score$^-$ and the right-end is score$^+$. Top-2 dominating results retrieved is thus ω_1 and ω_4.

Efficient Computation of Dominating Probabilities. $P(u \prec V)$ denotes the probability that an instance $u \in U$ dominates an uncertain object V; that is, the sum of the probabilities of the instances in V which are dominated by u. For instance regarding the example in Figure 5.2, $P(o_3 \prec B) = 1/3$ (recall each instance takes the probability 1/3 to occur).

Let $P_{=\ell}(u)$ denote the probability that an instance $u \in U$ dominates ℓ other objects. $\Omega_\ell^{\mathcal{U}-U}(u)$ denotes the subset of possible worlds in $\prod_{V \in \mathcal{U}-U} V$ in each of which u dominates exactly ℓ instances. Clearly, $P_{=\ell}(u) = \sum_{W \in \Omega_\ell^{\mathcal{U}-U}(u)} P(W)$.

Example 5.4. *Regarding Figure 5.2, let $U = E$, and $\ell = 2$. Then, $\Omega_\ell^{\mathcal{U}-U}(e_1) = \{(o_3, b_1), (o_3, b_2), (o_3, b_3)\}$. $P_{=2}(\{e_1\}) = 1/3 * 1/3 + 1/3 * 1/3 + 1/3 * 1/3 = 1/3$.*

We can immediately verify that (5.1) can be re-written as follows.

$$P_{=\ell}(U) = \sum_{u \in U} P(u) P_{=\ell}(u). \tag{5.3}$$

Based on (5.2) and (5.3), $P_{\geq \ell}$ can be rewritten as:

$$P_{\geq \ell}(U) = \sum_{u \in U} (P(u) \cdot (1 - \sum_{i=0}^{\ell-1} P_{=i}(u))). \tag{5.4}$$

According to Equation 5.3, a key to compute $pscore(U)$ and $P_{=\ell}(U)$ is to efficiently compute $P_{=\ell}(u)$ for each $u \in U$. Suppose that we already computed $P(u \prec V)$ for every $V \in \mathcal{U} - U$. The dynamic programming based techniques in [YLKS08] can be immediately used to compute $P_{=\ell}(u)$ ($\forall u \in U$) with time complexity $O(|\mathcal{U} - U| \times \ell)$ for a given u. Assume that uncertain objects in $\mathcal{U} - U$ are represented by $\{V_i : 1 \leq$

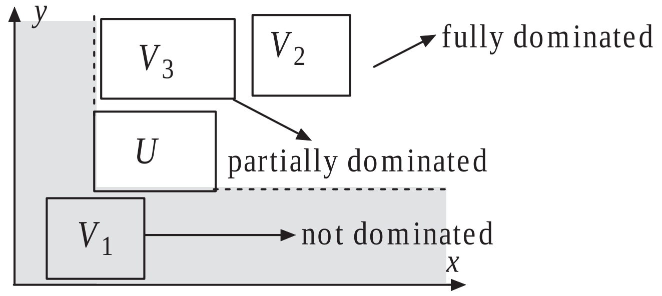


Figure 5.5: Dominating Relationships.

$i \leq n-1\}$; note that objects in $\mathcal{U} - U$ can follow any order. We use p_i to denote $P(u \prec V_i)$. For $0 \leq n_1 \leq n_2$, let P_{n_1,n_2} denote the probability that u exactly dominates n_1 objects from the first n_2 objects of $\mathcal{U} - U$. It is shown [YLKS08] that $\forall 0 \leq i \leq j$ ($P_{0,0} = 1$),

$$\begin{aligned} P_{0,j} &= P_{0,j-1} \cdot (1-p_j) = \Pi_{k=1}^{j}(1-p_k) \\ P_{i,j} &= p_i \cdot P_{i-1,j-1} + (1-p_i) \cdot P_{i,j-1} \end{aligned} \tag{5.5}$$

Let $FD(u)$ denote the set of objects fully dominated by u; that is, $\forall U \in FD(u)$, $P(u \prec U) = 1$. Let $PD(u)$ denote the set of objects partially dominated by u. It can be immediately verified that:

$$P_{=\ell}(u) = P_{=(\ell - |FD(u)|)}|_{PD(u)}(u \prec PD(u)). \tag{5.6}$$

Here, $P_{=(\ell-|FD(u)|)}(u)|_{PD(u)}$ denotes the probability that u dominates exactly $(\ell - |FD(u)|)$ objects in $PD(u)$ since the probability for u to dominate each object in $FD(u)$ is always 1. Consequently, in our techniques for each u we apply the dynamic programming technique on objects in $PD(u)$ only. Whenever there is no ambiguity, $P_{=l}(u)$ (or $P_{\geq l}(u)$), thereafter, always refers to the dominating probability against $PD(u)$ and $l = \ell - |FD(u)|$ where $\ell > |FD(u)|$ since all objects in $FD(u)$ are dominated by u with the probability 1.

Example 5.5. *Regarding Figure 5.2,* $p_1 = P(e_1 \prec O) = 1/3$, *and* $p_2 = P(e_1 \prec B) = 1$. *By the above dynamic programming based algorithm,* $P_{0,1} = 1 - p_1 = 2/3$, $P_{0,2} =$

$P_{0,1} * (1 - p_2) = 0$, $P_{1,1} = p_1 = 1/3$, $P_{1,2} = P_{0,1} * p_2 + P_{1,1} * (1 - p_2) = 2/3$. Thus, $P_{=1}(e_1) = 2/3$.

5.1.3 Challenges

1. A solution to PtopkQ highly depends on the probability distribution of objects even if spatial locations of the instances are fixed.

   Example 5.6. *Regarding the example of Figure 5.2, if we fix the spatial locations of these 9 instances but change the probability of instances from O'neal as follows, $P(o_1) = 1/6$, $P(o_2) = 1/6$ and $P(o_3) = 2/3$. The occurrence probability of every other instance remains $1/3$. Then, we can immediately verify that regarding $q = 2/3$, $pscore(O) = 1$, $pscore(E) = 2$ and $pscore(B) = 0$. In this case, the top-1 dominating query retrieves Brand instead of O'neal (the top-1 result in Example 5.2).*

2. Techniques developed solely on aggregate information cannot provide a correct solution to PtopkQ. It should be very straightforward to construct two different scenarios with the same aggregate information as depicted in Figure 5.1 such that they lead to different solutions towards PtopkQ.

3. The computation of $pscore(U)$ for an uncertain object U takes $O(|U| \times pscore(U) \times |PD(U)|)$ time as shown above. Trivially computing $pscore(U)$ for all $U \in \mathcal{U}$ and then choosing k objects with the highest $pscore$ values is computationally very expensive and slow.

5.2 Framework

Our exact and randomized algorithms both follow the threshold-based paradigm by using a combination of two thresholds based on q and $pscores$, respectively, to efficiently prune away objects not in PtopkQ as early as possible. Below, Algorithm 5.1 is an outline of the framework to be adopted in the exact and randomized algorithms. It follows three steps, pre-ordering, initial computation and final computation.

Algorithm 5.1 Exact Algorithm

Step 1: Pre-ordering. For all uncertain objects $\mathcal{U}$, generate an ordered list $\overrightarrow{\mathcal{U}}$ of $\mathcal{U}$.

Step 2: Initial Computation. Choose the first k objects $\{U_i : 1 \leq i \leq k\}$ in $\overrightarrow{\mathcal{U}}$ and compute their *pscore* (for exact algorithm) or *pscore*^{r} (for randomized algorithm) values.

Step 3: Final Computation. Determine the solution of PtopkQ in a "level-by-level" fashion.

Using $\overrightarrow{\mathcal{U}}$ resulted in Step 1, score values for the first k objects are computed in Step 2. Such values serve as thresholds in Step 3.

5.2.1 Data Structures

In the exact and randomized algorithms, we maintain an aR-tree on centroids to run CBT algorithm [YM07] as preprocessing (Step 1). We also maintain an aR-tree on the MBBs of uncertain objects to speed-up our pruning techniques at the object level.

Moreover, in the exact algorithm, for each object U, we build a local data structure, aR-tree, to organize its instances to efficiently support a level-by-level pruning computation in Step 3. However, the randomized algorithm indexes the sampled instances of each uncertain object using a novel data structure *gCaR-tree* for efficiency.

5.2.2 Monotonic Property

The following monotonic property will be effectively used to terminate our algorithm as early as possible. It immediately derives from Equation(5.2).

Monotonic Property: For an uncertain object U and two integers ℓ_1 and ℓ_2, if $\ell_1 \geq \ell_2$, $P_{\geq \ell_1}(U) \leq P_{\geq \ell_2}(U)$.

5.2.3 Efficient Level-by-level Computation

In the exact algorithm, for each uncertain object U in $\mathcal{U}$, instances in U are indexed using an aR-tree. Suppose that $E \in U$ is at the ith level of the aR-tree. Let $L_i(U)$ denote the set of entries in the ith level of local aR-tree of U. The equation (5.4) can be rewritten as:

$$P_{\geq l}(U) = \sum_{E \in L_i(U)} P_{\geq l}(E) \tag{5.7}$$

It will be too expensive to compute $P_{\geq l}(E)$ in our level-by-level computation. Instead, we use upper-bound techniques to bound $P_{\geq l}(E)$ for efficiency.

Let $(\mathcal{U} - U)_i$ denote the objects in $\mathcal{U} - U$ with the following modification regarding level i. For each object $V \in \mathcal{U} - U$ and each entry E_V at the ith level of the local aR-tree of V, we move all the instances contained by E_V to the upper-right corner μ_{E_V} of E_V. Let $\imath_E$ denote the lower-left corner of E. Let $P_{\geq \lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ denote the probability that $\imath_E$ dominates at least λ objects in $(\mathcal{U} - U)_i$.

Theorem 5.1. $P_{\geq \lambda}(E) \leq P_{\geq \lambda}(\imath_E \prec (\mathcal{U} - U)_i) \sum_{u \in E} P(u)$.

Proof. It can be immediately verified that for each possible world in the original case where an instance u from E dominates at least λ instances from different objects, its corresponding instance as modified above retains such a property. □ □

It is immediate that an application of the dynamic programming based algorithm in Section 5.1.2 leads to the time complexity $O(m_1 \times C \times \lambda)$ to compute $P_{\geq l}(E)$ where m_1 is the number of instances in E and C is the average cost to compute dominating probability between an instance and an object, while the computation of the upper-bound in Theorem 5.1 only takes $O(\lambda \times m_2)$ time where m_2 is the number of entries partially dominated by E. Clearly, m_2 is much smaller than C.

Example 5.7. *In Figure 5.4, assume that we want to compute $P_{\geq\lambda}(U)$. Theorem 5.1 states that we can get an upper-bound of $P_{\geq\lambda}(U)$ at the root level of local aR-trees of objects. Let $\imath_3$ be the lower left corner of the MBB of U_3 and μ_i (for $1 \leq i \leq 9$) be the upper right corner of U_i.*

Then, $(\mathcal{U} - U_3)_1 = \{\mu_i | 1 \leq i \leq 9 \ \& \ i \neq 3 \ \& \ P(\mu_i) = 1\}$. Theorem 5.1 states that $P_{\geq\lambda}(U) \leq P_{\geq\lambda}(\imath_3 \prec (\mathcal{U} - U_3)_1)$ since $\sum_{u \in U_3} P(u) = 1$.

5.3 Exact Algorithm

We present detailed techniques developed based on the framework in Section 6.2. The first step and the second step are quite straightforward and mainly based on the techniques in [YM07, YLKS08]. The third step is the most important step in Algorithm 5.1 to prevent as many objects as possible from an exact computation of *pscore*; novel, effective, efficient pruning techniques are developed.

5.3.1 Step 1: Pre-ordering Objects

Step 1 aims to generate such an access order so that the maximal possible threshold value regarding *pscore* can be reached as soon as possible. Clearly, the maximum possible threshold value regarding *pscore* should be the minimum value of the *pscores* of the top-k objects. Nevertheless, this is infeasible to achieve without conducting an exact computation of PtopkQ. The following heuristic is developed to resolve this.

The centroid $\omega(U)$ ($\forall U \in \mathcal{U}$) is used to approximately represent the probabilistic distribution of an uncertain object U with the aim to use $score(\omega(U))$ to approximately reflect the rank of $pscore(U)$. Note that it is quite expensive to compute $score(\omega(U))$ for each object U. Instead, we apply the CBT algorithm (briefly introduced in section 5.1.2) to generate an approximately ordered list $\overrightarrow{\mathcal{U}}$ as follows.

In $\overrightarrow{\mathcal{U}}$, we keep the scored entries of the aR-tree of centroids, generated by CBT; that is, the entries pruned by CBT or the entries remained in the job queue once it terminates (as described in Section 5.1.2). Then, we sort entries in $\overrightarrow{\mathcal{U}}$ non-increasingly according to their accompanied $score^+$ values. When a centroid $\omega(U)$ and the intermediate entry E have the same $score^+$ value, we always rank $\omega(U)$ before E in $\overrightarrow{\mathcal{U}}$. Then, if two $score^+$ values from two centroids are the same, we always rank a centroid with the exact *score* value higher. In other cases, entries with the same score are ranked randomly among them. Note that in an entry, each contained centroid $\omega(U)$ corresponds to the object U; we use U to replace $\omega(U)$ in $\overrightarrow{\mathcal{U}}$.

Example 5.8. *Regarding the example in Figure 5.3 and Figure 5.4, Figure 5.3 shows the centroids of uncertain objects in Figure 5.4. As shown in Example 5.3 when* $k = 2$,

$$\{\omega_1.[6,6], \omega_4.[4,4], \omega_2.[3,3], \omega_3.[3,3], \omega_5.[0,3], \omega_6.[0,0]\},$$

remain in job queue, while $E_3.[0,2]$ *is pruned by CBT. Consider that* ω_i *(for* $1 \leq i \leq 9$*) corresponds to the uncertain object* U_i*. Therefore,* $\overrightarrow{\mathcal{U}} = \{U_1, U_4, U_2, U_3, U_5,\ E_3, U_6\}$ *when* $k = 2$*. Their* $score^+$ *values are* 6, 4, 3, 3, 3, 2, *and* 0, *respectively.*

5.3.2 Step 2: Initial Computation

Our algorithm to calculate the *pscore*s for each U of the first k objects in $\overrightarrow{\mathcal{U}}$ is outlined below in Algorithm 5.2.

Algorithm 5.2 Calculate *pscore*

```
Step 2.1: Traverse the $aR$-tree of objects' MBBs to obtain the number of objects that $U$
    fully dominates $|FD(U)|$, and the set $PD(U)$ of objects that $U$ partially dominates.
Step 2.2: Do a synchronous traversal [BKS93, PMT99] of the local $aR$-tree of $U$ against
    the local $aR$-trees of the objects in $PD(U)$ to calculate $P(u \prec V)$ for each $V \in PD(U)$ and each instance $u \in U$.
Step 2.3: Calculate the $pscore(U)$.
```

We conduct step 2.1 by window query techniques [Gut84] by using $\imath_U$ to get all objects that U dominates (fully or partially) and then use μ_U to check the full dominance.

We conduct Step 2.2 by the well known synchronous traversal paradigms [BKS93, PMT99] to compute $P(u \prec V)$ ($\forall u \in U$ and $\forall V \in PD(U)$) since the synchronous traversal paradigm has been shown effective in join computation. Moreover, [YM07] shows that on average the synchronous traversal strategy is the most cost effective way to count the dominance relationships. Finally, our techniques can be extended to cover any traversal strategies.

Note $P_{\geq l}(U) = \sum_{u \in U} P(u) P_{\geq l}(u)$. In Step 2.3, to calculate $P_{\geq l}(U)$ we apply the dynamic programming based algorithm in Section 5.1.2 to calculate $P_{\geq l}(u)$ ($\forall u \in U$) **restricted to the objects in** $PD(U)$. Based on the monotonic property in Section 5.2.2, when $P_{\geq l}(U) \geq q$ and $P_{\geq (l+1)}(U) < q$, the computation stops and $(l + |FD(U)|)$ is the *pscore* for U. To avoid any redundant computation, we conduct the calculation in Equation (5.4) iteratively from $l = 0$. After the completion of calculation of $P_{\geq l}(u)$ for each $u \in U$ for the current l, we examine if $P_{\geq l}(U) \geq q$ to determine whether we should stop a further calculation of such probability. We can immediately verify that the time complexity of Step 2.3 is $O(l \times |PD(U)| \times |U|)$ for each U.

5.3.3 Step 3: Final Computation

The final computation is conducted by **bounding-pruning-refining**. This will be based on a threshold of *pscore* and the given confidence q. Clearly, the best available threshold of *pscore* is the minimum value, denoted by λ_k, of *pscore*s of the current top-k objects. To pursue efficiency, for each remaining U the Step 3 will be conducted level-by-level in a synchronous traversal fashion among the local aR-trees of U and the objects in $PD(U)$;[3] nevertheless, our techniques can be extended to any traversal strategies. Our algorithm for Step 3 is outlined in Algorithm 5.3.

While Steps 3.1 and 3.3 are relatively straightforward, Step 3.2 is critical in Algorithm 5.3; it can significantly speed-up the algorithm by avoiding as many objects as possible to enter into the expensive Step 3.3; our experiment results demonstrate that our pruning techniques can speed-up the computation by orders of magnitude. We show the basic idea of our algorithm of Step 3.2 in Example 5.9. Suppose that E is an entry, at the ith level, of the local aR-tree of U, let $PD(E)$ denote the set of *entries* at the ith level of the local aR-trees of other objects, which are partially dominated by E. $\#obj(PD(E))$ denotes the number of distinct objects containing the entries in $PD(E)$, while $FD(E)$ denotes the set of *objects* fully dominated by E.

[3]Note that if local aR-trees have different height, the one that reaches the bottom level first will stay at the bottom, while others traverse down to the lower levels.

Algorithm 5.3 Final Computation

```
$T_k$ := {the first $k$ objects from $\overrightarrow{\mathcal{U}}$}; $\overrightarrow{\mathcal{U}} := \overrightarrow{\mathcal{U}} - T_k$;
WHILE $\overrightarrow{\mathcal{U}} \neq \emptyset$ DO
    Step 3.1 - Pruning at Object Level: Dequeue the first entry $E$ from $\overrightarrow{\mathcal{U}}$; Use window
        queries to check if objects in $E$ can be completely pruned away - if not, then go to
        Step 3.2.
    Step 3.2 - Level-by-Level Pruning: For each remaining $U$, do a level-by-level syn-
        chronous traversal among the local $aR$-tree of $U$ and the local $aR$-trees of the
        objects in $PD(U)$ to conduct a level-by-level pruning.
    Step 3.3 - Compute pscore: For each remaining object $U$ after Step 3.2,
        • calculate the $pscore(U)$;
        • if $pscore(U) > \lambda_k$, then replace an object $V$ in $T_k$ with $pscore(V) = \lambda_k$ by
          $U$, and Update $\gamma_k$.
ENDWHILE
Return $T_k$.
```

Example 5.9. *In Figure 5.6, the* 3 *local aR-trees of* U_1*,* U_2*, and* U_3 *have* 3 *levels, respectively, with one intermediate level* E_j *(*$\forall 1 \leq j \leq 9$*). Assume that* $\lambda_k = 1$ *and Step 3.2 is conducted against* U_1*.*

Note that $PD(U_1) = \{U_2, U_3\}$ *and* $FD(U_1) = \emptyset$*. As* $pscore^{+}(U_1) = |PD(U_1)| + |FD(U_1)| \geq \lambda_k$*, we expand* U_1*,* U_2*, and* U_3 *synchronously to the next level. The following is immediate where each* E_j *(for* $1 \leq j \leq 9$*) is at level* 2*.*

- $PD(E_1) = \{U_2.(E_5, E_6)\}$[4] *and* $FD(E_1) = \{U_3\}$*. Note that* $\#obj(PD(E_1)) = 1$*.*
- $PD(E_2) = \{U_3.(E_9)\}$ *and* $FD(E_2) = \emptyset$*. Note that* $\#obj$ $(PD(E_2)) = 1$*.*

[4]Note that E_6 is fully dominated by E_1; consequently we no longer need to expand E_6 regarding E_1 but just add $P(E_6)$ to calculate the probabilities and scores of the children of E_1.

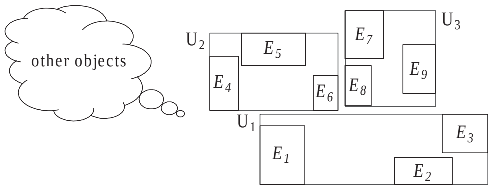


Figure 5.6: Level-by-level Computation.

- $PD(E_3) = \emptyset$ *and* $FD(E_3) = \emptyset$.

Since $pscore^+(E_3)(\triangleq \#obj(PD(E_3)) + |FD(E_3)|) = 0$ *(*$< \lambda_k$*), we can exclude* E_3 *from a further consideration. We only need to check* E_1 *and* E_2 *by the following bounding-pruning techniques to determine whether or not they need to be expanded to the next level.*

The key in Step 3 is to develop efficient and effective bounding-pruning techniques for pruning purposes. They will be conducted based on the following two principles.

1. **probability-based:** Efficiently and effectively computing an upper-bound $P^{upper}_{\geq\lambda_k}(U)$ of $P_{\geq\lambda_k}(U)$ so that U can be pruned if $P^{upper}_{\geq\lambda_k}(U) \leq q$.

2. **score-based:** Efficiently and effectively computing a $pscore^+(U)$ such that U can be pruned if $pscore^+(U) < \lambda_k$.

Efficient and Effective Bounding Techniques

In Theorem 5.1, for each entry E, we use $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$, multiplied by $\sum_{u\in E} P(u)$, as an upper bound of $P_{\geq\lambda}(E)$. This takes $O(\lambda_k \times |PD(E)|)$ time for each entry E. To further speed-up the computation, the following two upper-bounds of $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ are developed; they reduce the costs from $O(\lambda_k \times |PD(E)|)$ to $O(|PD(E)|)$. This is significant when λ_k is large.

1. Chernoff-Hoeffding Bound based Upper-bound. For an uncertain object V and an instance u in another uncertain object U, we can regard the event that u dominates V as a random variable. Consequently, we can employ the probabilistic bounds to compute the upper bound of the pscore of an uncertain object, which is very time efficient. Due to the independence assumption, we apply a strong version of Chernoff-Hoeffding Bound [DP98] in the chapter.

Chernoff-Hoeffding Bound [DP98]. Let X_1, X_2, X_3, ..., X_n be independent random variables with values in $[0,1]$, $X = \sum_{i=1}^{n} X_i$ and $\epsilon > 0$. Then,

$$P(X > (1+\epsilon)E(X)) < \exp^{-E(X)\epsilon^2/3} \tag{5.8}$$

Recall $\imath_E$ is the lower-left corner of an entry E. $\imath_E$ partially dominates l objects V_1, V_2, ... V_l. Since each V_i $(1 \leq i \leq l)$ is an uncertain object, the probability of $\imath_E$ dominating a V_i can be treated as the expected value of the following random variable.

$$X_{V_i} = \begin{cases} 1 & \text{if } \imath_E \text{ dominates one instance of } V_i \\ 0 & \text{otherwise.} \end{cases} \tag{5.9}$$

We can view the number of objects, dominated by $\imath_E$, as the sum of following random variables.

$$X_{\imath_E} = X_{V_1} + X_{V_2} + ... + X_{V_l} \tag{5.10}$$

Clearly, $E(X_{V_i}) = P(\imath_E \prec V_i)$, and $E(X_{\imath_E}) = \sum_{i=1}^{l} P(\imath_E \prec V_i)$. Since all V_is are mutually independent, we can apply the above Chernoff-Hoeffding bound with $\epsilon = \frac{(\gamma - E(X_{\imath_E}))}{E(X_{\imath_E})}$ to get Lemma 5.1, where $\gamma = \gamma_k - |FD(\imath_E)|$ and $|FD(\imath_E)|$ is the number of objects fully dominated by $\imath_E$.

Lemma 5.1. *If* $E(X_{\imath_E}) < \gamma$, *then* $P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i) \leq \exp^{-\frac{(\gamma - E(X_{\imath_E}))^2}{3E(X_{\imath_E})}}$.

In our pruning technique, we will use $\exp^{-\frac{(\gamma - E(X_{\imath_E}))^2}{3E(X_{\imath_E})}}$ as an upper-bound of $P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i)$. This will reduce the complexity of calculation from $O(\gamma \times l)$ to $O(l)$ when $\gamma > E(X_{\imath_E})$. This is significant when γ is large. Below, we present another upper-bound estimation of $P_{\geq\gamma}(\imath_E)$ when γ is relatively small — $\gamma \leq E(X_{\imath_E})$; in this case, Chernoff-Hoeffding Bound does not yield interesting results.

2. ***Bisection-based Upper-bound.*** Due to the above limitation when applying the Chernoff-Hoeffding Bound based Upper-Bound, we further develop a more general Upper-Bound called Bisection-Based Upper-bound. Following theorem is the key to obtain this upper bound. Without loss of generality, suppose that the l objects, partially dominated by the lower-left corner $\imath_E$ of an entry E, are sub-indexed such that $P(\imath_E \prec U_i) \leq P(\imath_E \prec U_j)$ if $i < j$. Let $p_i = P(\imath_E \prec U_i)$ for $1 \leq i \leq l$.

Theorem 5.2. *Suppose that we replace p_i by p_i^* for $1 \leq i \leq l$ such that $p_i \leq p_i^*$. Then, the probability that u dominates at least λ objects (for $1 \leq \lambda \leq l$) regarding $\{p_i : 1 \leq i \leq l\}$ is not greater than that regarding $\{p_i^* : 1 \leq i \leq l\}$.*

Theorem 5.2 is quite intuitive, but the proof is lengthy. Please refer to the appendix for the detailed proof.

Now, we can divide the probabilities of those partially dominated objects (by $\imath_E$) into two groups $\mathcal{G}_1 = \{p_1, p_2, ..., p_j\}$ and $\mathcal{G}_2 = \{p_{j+1}, p_{j+2}, ...p_l\}$ such that we replace each probability value in $\mathcal{G}_1$ by p_j and replace each probability value in $\mathcal{G}_2$ by p_l. The following Lemma is immediate.

Lemma 5.2. *Without loss of generality, we assume that $j \leq (n - j)$, let $y_0 =$*

$max\{0, \lambda - l + j\}$

$$P_{\geq\gamma}(\imath_E \prec (\mathcal{U} - U)_i) \leq \sum_{y=y_0}^{j} C_j^y p_j^y (1-p_j)^{j-y} \times \left(\sum_{x=\lambda-y}^{l-j} C_{l-j}^x p_l^x (1-p_l)^{l-j-x}\right) \quad (5.11)$$

Proof. Suppose that the instance u dominates l objects with the probabilities, $\overbrace{p_j, p_j \cdots, p_j}^{j}, \overbrace{p_l, p_l, \cdots, p_l}^{l-j}$. It can be immediately verified that the probability that $\imath_E$ dominates at least λ objects among these l objects is as what is stated on the right hand-side of the inequality of (5.11). The lemma immediately follows from Theorem 5.2. □ □

Lemma 5.2 states that we can bisect the set of partially dominated objects into two groups such that in each group, we use the largest probability value as a representative. Then, we use the right-side part of the inequality in (5.11) as an upper-bound. Clearly, it can be calculated in $O(l)$ time if we accumulatively compute the part, $\sum_{x=\lambda-y}^{l-j} C_{l-j}^x p_l^x (1-p_l)^{l-j-x}$, from the tail.

The key to deliver a good upper-bound is to choose a p_j such that the value of upper-bound can be minimized. This problem can be trivially solved in time $O(l^2)$ by enumerating all possible cases; nevertheless, such costs are even more expensive than the costs $O(\lambda \times l)$ of the dynamic programming based algorithm to produce the exact probability value.

In our computation, we choose the median to divide the set into two groups. It is clear that the median can be calculated in $O(l)$ time [CLRS01]. Therefore, the whole computation of upper-bound can be executed in time $O(l)$.

Remark 1. *It seems hard to find an efficient algorithm with costs lower than $O(\lambda l)$ to divide l probability values into more than 2 groups; consequently we settle for a bisection. The bisection-based upper-bound can also be used in case when*

$\gamma > E(X_{\imath_E})$. However, our experiments, in Section 6.5, demonstrate that the above Chernoff-Hoeffding bound based upper-bound is tighter than the bisection-based upper-bound. Therefore, in our implementation we only use the Chernoff-Hoeffding bound for the case where $\gamma > E(X_{\imath_E})$. These two bounds will be used to calculate the upper-bounds of $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$ in our level-by-level computation.

We also examined Markov's inequality [Gol01] and Chebyshev's inequality [Gol01]; the upper-bounds generated by them are not as tight as the above two upper-bounds.

3. Utilizing Existing Computation Results. Below we show two upper-bounds by utilizing the existing computation results. One is dominating probability based, while another is *pscore* based.

Theorem 5.3. *Suppose that u is a point (or an instance of U_1) and u fully dominates an uncertain object, say, U_2. Then, $P_{\geq\gamma}(u) \geq P_{\geq\gamma}(U_2)$ ($\forall\gamma \geq 1$).*

The proof of Theorem 5.3 is quite lengthy and we leave it to Appendix.

Note that Theorem 5.3 will be used to prune away objects, fully dominated by u, if $P_{\geq\lambda}(u) < q$. The following Theorem is immediate.

Theorem 5.4. *Suppose that a point u (partially or fully) dominates λ' objects in total, and u dominates the lower-left corner of the MBB of an entry E of the local aR-tree of an object U at the level i. Then, $pscore^{+}(E) \leq \lambda'$.*

Note that in Theorem 5.4, level $i = 1$ means an object.

Effective Pruning Rules

The pruning rules below can be immediately verified from the definitions; thus we omit the proofs.

Pruning Rule 5.1. Score-Based: *$\forall U$, if $pscore^+(U) \leq \lambda_k$, then U can be excluded from the solution of PtopkQ.*

Let $L_i^+(U)$ denote the subset of entries of $L_i(U)$ with the property that $\forall E \in L_i^+(U)$, the captured $P_{\geq \lambda_k}^{upper}(E) \neq 0$. Based on equation (5.7), the following pruning rule is immediate.

Pruning Rule 5.2. Level's Probability-based: *Suppose that $\sum_{E \in L_i^+(U)} P_{\geq \lambda_k}(E) \leq q$. Then, U can be excluded from the solution of PtopkQ.*

Note that when $i = 1$, $L_i^+(U)$ in Pruning Rule 5.2 only contains the root entry: U.

In our computation if instances or entries in U are found with 0 probability to dominate λ_k objects, we mark and exclude them in further computation. For each entry E of a local aR-tree, let $\mathcal{I}_E^+$ denote the set of instances each of which is not yet detected with 0 probability to dominate at least λ_k objects, and $P(\mathcal{I}_E^+)$ denotes the sum of probabilities of instances in $\mathcal{I}_E^+$. The Pruning Rule 5.3 below is also immediate if we make the upper-bound of probability for an instance in $\mathcal{I}_E^+$ to dominate at least λ_k objects be 1.

Pruning Rule 5.3. Drilling-down based: *At the level i (for an i), if $\sum_{E \in L_i^+(U)} P(\mathcal{I}_E^+) < q$, then U can be excluded from the solution of PtopkQ.*

Pruning rules 5.2 and 5.3 are fundemental to a level-by-level computation (details in Section 5.3.3).

Remark 2. *Note that in our level-by-level algorithm, an $\mathcal{I}_E^+$ may change when levels progress down. For instance, regarding the example in Figures 5.7 and 5.8, E_{13} and E_{15} are initially detected with $P_{\geq \lambda_k}(E_{15}) = 0$ and $P_{\geq \lambda_k}(E_{13}) = 0$ because they are fully dominated by one point that (partially or fully) dominates no more*

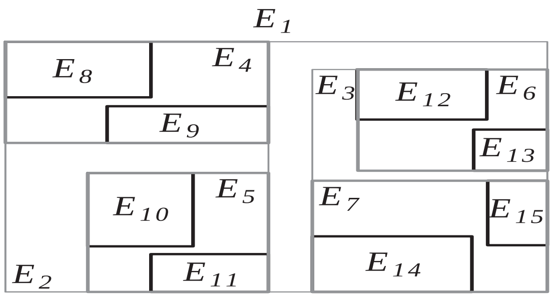


Figure 5.7: Entry Distribution

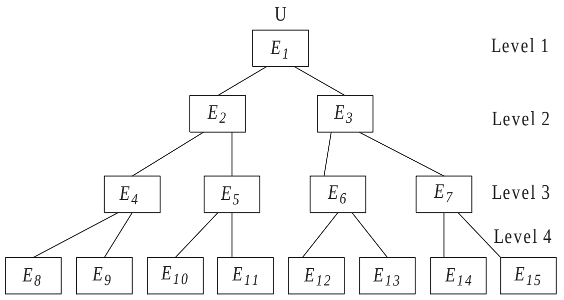


Figure 5.8: Tree Structure Map

than the current λ_k objects (formally stated in Theorem 5.4); consequently, $\mathcal{I}^+_{E_6}$ contains the instances contained by E_{12}. Nevertheless, once progress to level 2, we may find that the total number of objects (fully or partially) dominated by E_6 is less than threshold λ_k; consequently, in $\mathcal{I}^+_{E_6}$ we replace E_{12} by $\varnothing$. Thus $\mathcal{I}^+_{E_6}$ is empty.

Algorithm Details

Step 3.1. Objects corresponding to the centroids in an entry E of the aR-tree on centroids may be spread to different entries of the aR-tree on object MBBs.

Example 5.10. *Regarding the centroids ω_1, ω_2, and ω_3 in Figures 5.3, their corresponding objects U_1, U_2, and U_3 are spread to 2 entries in the local aR-tree on object MBBs.*

In each entry E of the local aR-tree on object MBBs, we record the lower left corner of the MBB encompassing the objects that correspond to the contained centroids in E, denoted by $\jmath_E$. Note that $\jmath_E$ is not the lower left corner of E. Below is the algorithm presented in Algorithm 5.4.

To compute $pscore^+(E)$ - an upper-bound of the maximum number of objects (partially or fully) dominated by an object in E, we use window query techniques by the "half-open" window with $\jmath_E$ as the lower-left corner to probe the aR-tree on MBBs of objects and then count the number of objects overlapping with the

Algorithm 5.4 Step 3.1
Description:

```
1:  get $pscore^+(E)$;
2:  if $pscore^+(E) \leq \lambda_k$ (Pruning Rule 5.1) then
3:      prune other objects dominated by $\jmath_E$
4:  else
5:      if $E$ is an object $U$ then
6:          record $PD(U)$ and goto Step 3.2;
7:      else
8:          for each child $E'$ of $E$ do
9:              call Algorithm 5.4 regarding $E'$;
10:         end for
11:     end if
12: end if
```

window as $pscore^+(E)$.

If the condition in line 2 holds, then objects corresponding to the centroids in E will be excluded from a further consideration. In this case, we can prune other objects by $\imath_E$ by using the above window to probe the aR-tree of objects to get the objects fully dominated by $\jmath_E$. These objects will be removed from $\overrightarrow{\mathcal{U}}$ or from an entry in $\overrightarrow{\mathcal{U}}$. Note that when objects removed from an entry E of $\overrightarrow{\mathcal{U}}$, we need to update $\jmath_E$ and the corresponding information in its descendants. Moreover, if an entry in the aR-tree of objects is detected to be fully dominated by $\jmath_E$, then it is marked so that the entry can be skipped when another $\jmath_{E'}$ is used to prune away objects.

Example 5.11. *In Step 3.1, suppose the current λ_k is 3. When the entry containing ω_7, ω_8, and ω_9 is selected, we use the recorded lower-left corner (with this entry) of the MBB* *of objects U_7, U_8, and U_9 to do the window query on the aR-tree of objects. The window query does not intersect any object. Consequently, the entry containing ω_7, ω_8, and ω_9 will be removed from candidates, and U_7, U_8, and U_9 are excluded from the candidates of PtopkQ.*

Remark 3. *At the object level, we also use Pruning Rule 5.3 to check (line 2 of Algorithm 5.4) if an object should be removed from the candidates of PtopkQ.*

Step 3.2. For each remaining object U, we synchronously traverse the local aR-trees of U and objects in $PD(U)$ level-by-level such that at each internal level i, we conduct the following two substeps.

Step 3.2a. Use Pruning Rule 5.3 to check if U should be removed. If U cannot be removed, then go to Step 3.2b.

Step 3.2b. For each $E \in L_i^+(U)$, we compute $PD(E)$ and $|FD(E)|$. Then, based on Theorem 5.1 we use Chernoff-Hoeffding bound based upper bound or Bisection based upper bound to bound $P_{\geq\lambda}(\imath_E \prec (\mathcal{U} - U)_i)$, which is multiplied by $\sum_{u \in E} P(u)$ to give an upper bound $P_{\geq\lambda_k}^{upper}(E)$ of $P_{\geq\lambda_k}(E)$. Then, we use Pruning Rule 5.2 to check if U should be excluded or goto the next level. Note that when applying Pruning Rule 5.2, we replace $P_{\geq\lambda_k}(E)$ by $\min\{P_{\geq\lambda_k}^{upper}(E), P(\mathcal{I}_E^+)\}$.

To efficiently execute Pruning Rule 5.3, for each entry E we record the summation $p^0(E)$ of occurrence probabilities of detected instances that have 0 probability to dominate at least λ_k objects. Once an entry E is detected to have every instance with 0 probability dominating at least λ_k objects, this information is propagated to all ancestors as follows if E is the first time, (i.e. $full(E) = 0$), detected. Let $full(E) = 1$ denote the situation that every instance in E has already been detected to be with 0 probability dominating at least λ_k objects.

Algorithm 5.5 Propagation to Ancestors

Description:

```
1: if $full(E) = 0$ then
2:     $full(E) = 1$; $p' := p^0(E)$; $p^0(E) := P(E)$;
3:     for each ancestor $E'$ of $E$ do
4:         if $full(E') = 0$ then
5:             $p^0(E') := p^0(E') + P(E) - p'$;
6:             if $P(E') = p^0(E')$ then
7:                 $full(E') = 1$
8:             end if
9:         else
10:            Terminate
11:        end if
12:    end for
13: end if
```

Example 5.12. *Regarding the example in Figures 5.7 and 5.8, suppose that E_{15} is detected to be fully dominated by a point that has zero probability to dominate at least λ_k objects. Then, $P_{\geq\lambda_k}(E_{15}) = 0$. Further suppose that each entry at the bottom level has instances with the total probability $1/8$. Thus, we record $full(E_{15}) = 1$, $P^0(E_{15}) = P^0(E_7) = P^0(E_3) = P^0(E_1) = 1/8$.*

Assume that another such point is found to fully dominate E_3. Then, update $full(E_3)$ to be 1, and $P^0(E_3) = 1/2$ and $P^0(E_1) = 1/2$. If we find the third such point that fully dominates E_{15}, the search of E_{15} will stop at E_3 since $full(E_3) = 1$.

Remark 4. *Once the lower-left corner $\imath_E$ of an entry E is detected to have 0 probability to dominate at least λ_k objects, we use window query techniques to check if entries from other objects are fully dominated by $\imath_E$. For any entry fully dominated* *by $\imath_E$, we apply Algorithm 5.5 to propagate to ancestors of the entry. Moreover, when an object is processed as a candidate in Step 3.2, we do not need to expand its entries E with $full(E) = 1$.*

Step 3.3. We use the dynamic programming method to calculate $pscore(U)$ as what is described in Step 2. Note that when an instance u is detected $P_{\geq \lambda_k}(u) < q$, we can apply Theorem 5.3; that is, we do window queries, in the same way as described in the above step, by excluding all objects fully dominated by u, and update $\overrightarrow{U}$ accordingly.

5.4 Randomized Algorithm

The basic idea of our randomized algorithm is to sample all possible worlds, $\prod_{i=1}^{n} U_i$ from $\mathcal{U} = \{U_i | 1 \leq i \leq n\}$, by a small number m of possible worlds $\mathcal{S}_i$ ($1 \leq i \leq m$), where each $\mathcal{S}_i$ consists of n instances - one instance per object. An instance u *homo-dominates* another instance v if u dominates v, and they are in one sample $\mathcal{S}_i$. Let $u_{i,j}$ denote an instance in sample S_i from object U_j. $pscore^r(u_{i,j})$ is defined as the number of instances in sample S_i that are dominated by $u_{i,j}$; that is, the number of instances homo-dominated by $u_{i,j}$. For $1 \leq j \leq n$, $pscore^r(U_j)$ is the $(q * m)$th largest in $\{pscore^r(u_{i,j}) | 1 \leq i \leq m\}$.

Example 5.13. *Regarding the example in Figure 5.9, suppose that $m = 8$, $k = 2$, and $q = 0.5$. A circled number j in object U_i means the sampled instance (from U_i) is in the sample j. The $pscore^r$ of object U_1 is 2. This is because that the samples 1, 2, 3 and 4 homo-dominate two other samples respectively (i.e. samples with the same sub-indexes) from U_2 and U_3, while samples 5, 6, 7, and 8 homo-dominate 1 sample, respectively.*

Similarly, we obtain that $pscore^r(U_2) = 1$ and $pscore^r(U_3) = 0$. Therefore, Algorithm 5.6 returns U_1 and U_2 as the top-k objects.

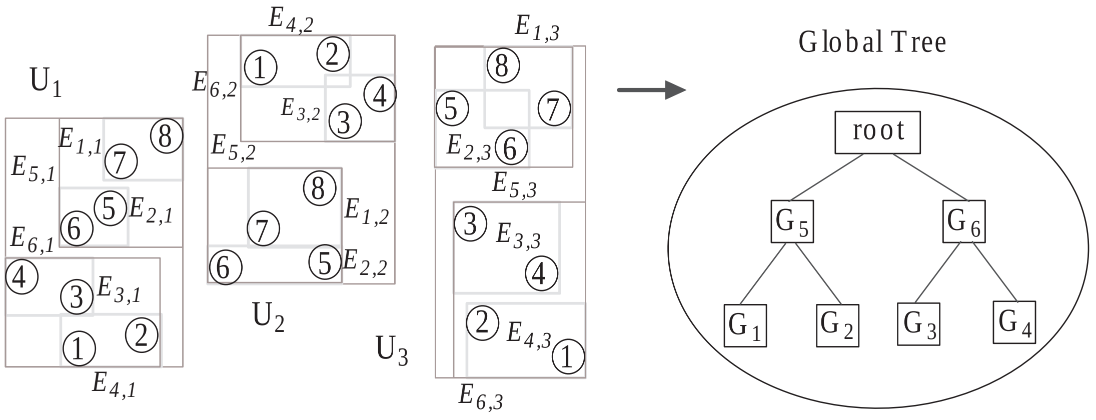


Figure 5.9: Samples

Below, Algorithm 5.6 outlines our randomized algorithm.

Algorithm 5.6 Randomized Algorithm

Input: $\{\mathcal{S}_i : 1 \le i \le m\}$; $0 < q \le 1$.

Output: $\mathcal{T}_k$: the k objects with the largest $pscore^r$.

Description:

```
1: $\mathcal{T}_k$ := Calculating-$pscore^r$ ($\{\mathcal{S}_i : 1 \le i \le m\}, q$);
2: return $\mathcal{T}_k$
```

In Algorithm 5.6, Calculating-$pscore^r$ $(\{\mathcal{S}_i : 1 \le i \le m\}, q)$ returns the k objects with the highest $pscore^r$s. A naive way of Calculating-$pscore^r$ is to compute the dominating number for each sampled instance in $\mathcal{S}_i$ for $1 \le i \le m$; consequently, we need to perform such computation m times if there are m samples. Our experiments demonstrate such a naive algorithm is very expensive, slow, and not scalable against m. Below, we present a novel, efficient algorithm for Calculating-$pscore^r$ with the aim to share the computation among samples and to effectively prune away objects. First, we show an accuracy guarantee of the algorithm.

5.4.1 Accuracy Guarantee

For each object U_j, the events whether in sample $\mathcal{S}_i$, the randomly selected instance $u_{i,j}$ dominates at least l other instances may be described by the following totally independent random variables.

$$X_{\geq l,i,j} = \begin{cases} 1 & \text{if } u_{i,j} \text{ dominates at least } l \text{ instances in } \mathcal{S}_i \\ 0 & \text{otherwise} \end{cases}$$

Clearly, $E(X_{\geq l,i,j}) = \sum_{u \in U_j} P(u) P_{\geq l}(u) = P_{\geq l}(U_j)$. Let

$$X_{\geq l,j} = \frac{\sum_{i=1}^{m} X_{\geq l,i,j}}{m}.$$

It is immediate that $E(X_{\geq l,j}) = P_{\geq l}(U_j)$. Theorem 6.6 immediately follows the Hoeffding's inequality [Gol01] (Theorem 5.6).

Theorem 5.5. *If $m = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ where $0 < \delta, \epsilon < 1$, then $P(|X_{\geq l,j} - P_{\geq l}(U_j)| \geq \epsilon) < \delta$.*

Theorem 5.6. Hoeffding's Inequality: *Suppose that Y_1, Y_2, ... , Y_m are independent random variables such that $0 \leq Y_i \leq 1$ for $1 \leq i \leq m$. Let $Y = \sum_{i=1}^{m} Y_i$. Then, we have that:*

$$P(Y - E(Y) \geq \epsilon m) \leq \exp^{(-2\epsilon^2 m)} \tag{5.12}$$
$$P(E(Y) - Y \geq \epsilon m) \leq \exp^{(-2\epsilon^2 m)}$$

Theorem 6.6 implies that $P_{\geq l}(U_j) - \epsilon \leq X_{\geq l,j} \leq P_{\geq l}(U_j) + \epsilon$ with confidence $1 - \delta$.

In our randomized algorithm — Algorithm 5.6, we use $X_{l,j}$ to approximately represent $P_{\geq l}(U_j)$; consequently, $(q * m)$th greatest number ($pscore^r(U_j)$) of homo dominated instances is used to approximately represent $pscore_q\ (U_j)$. Below is a theoretical guarantee of our randomized algorithm.

Lemma 5.3. *In Algorithm 5.6, suppose that we replace q by $(1-\epsilon)q$ in Algorithm 5.6, replace ϵ by ϵq in Theorem 6.6, and change m from $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ to $O(\frac{1}{\epsilon^2 q^2}\log\frac{n}{\delta})$. Then the top-$k$ objects retrieved by Algorithm 5.6 have the following properties with confidence $1-\delta$. For $1 \leq i \leq k$ ($\forall k \leq n$),*

Property 1: *the pscore^r of the top ith object U_i is not smaller than the pscore of the top ith object to PtopkQ (regarding q);*

Property 2: $P_{\geq pscore^r}(U_i) > (1-2\epsilon)q$.

Proof. It can be immediately verified that for each object U_i,

$$X_{pscore^r(U_i)+1,i} > (1-\epsilon)q. \tag{5.13}$$

Consequently, we have $P_{\geq pscore^r(U_i)+1}(U_i) \leq q$ with very small probability $\frac{\delta}{n}$ applying Theorem 6.6. The Property 1 immediate follows.

From Theorem 6.6, Property 2 immediately follows. □ □

Note that the sample size in Theorem 6.6 and Lemma 6.6 is irrelevant to the number of instances in an object; thus, the randomized algorithm has a potential to support the applications where a large number of instances is involved.

While Theorem 6.6 and Lemma 5.3 provide theoretical performance guarantee, our experiments demonstrate that Algorithm 5.6 is quite accurate when m is up to 1000 and q, instead of $(1-\epsilon)q$ used in Algorithm 5.6.

5.4.2 Efficient Algorithm

We present an efficient algorithm to execute Calculating-*pscore^r*. It follows the framework of 3 Steps in Section 6.2, Pre-ordering, Initial Computation, and Final Computation. We first present a novel data structure to replace local aR-trees.

***gC*aR-tree.** The sampled instances of each object are organized into an R-tree like structure, *gCaR*-tree (*Global Constrained* aR-tree). Different from a conventional R-tree, n gCaR-trees for n objects (one gCaR-tree per object) follow a global tree structure as follows.

> Corresponding to each node G_i in the global tree, for all $1 \leq j, j' \leq n$, entries $E_{i,j}$ and $E_{i,j'}$ of U_j and $U_{j'}$ contain the instances from the same samples, respectively. For example, in Figure 5.9, corresponding to G_2, $E_{2,1}$, $E_{2,2}$, and $E_{2,3}$ contain the sampled instances from the samples 5 and 6, respectively.

In a gCaR-tree, the number of sampled instances in each entry is also recorded. Given a global tree structure, we aim to minimize the sum of areas of gCaR-tree for all uncertain objects. It can be immediately shown that this optimization problem is NP-hard since a special case of the problem (i.e., when $n = 1$) is the area minimization problem of an R-tree which is NP-hard.

We build n gCaR-trees following the techniques of building an R-tree except that we enforce the constraint of a global tree structure as above. gCaR-trees have the advantage that in level-by-level computation, only the *homo-dominating* relationships among objects need to be checked. An entry E (fully or partially) *homo-dominates* another entry E' if E and E' correspond to the same entry in the global tree and E (fully or partially) dominates E'.[5] Consequently, for each entry E we only need to check one entry per object to determine if there is a homo-dominating relationship. Thus, the total costs to compute all entries, at the ith level of gCaR-trees, homo-dominated by an entry in U takes $O(n')$ where n' is the number of objects partially dominated by U. This is much lower than $O(N)$ in the exact algorithm where N is the number of total entries at the ith level.

[5]in Figure 5.9, $E_{6,1}$ fully homo-dominates $E_{6,2}$ and $E_{6,1}$ partially homo-dominates $E_{6,3}$.

However, such a global constraint may also bring a disadvantage — sizes of MBBs may be too large. Clearly, traversing gCaR-trees from a parent E to a child E' does not bring much extra geometric information if the MBB of E' has a similar size to that of E. To resolve this, we introduce a post-processing as follows while building gCaR-trees.

Post-processing a *gC*aR-tree. We enforce the constraint that for each group G_i,

$$\frac{area(G_i)}{area(G_i^p)} \leq \rho. \tag{5.14}$$

Here, $area(G_i)$ denotes the total area of the MBBs from n gCaR-trees corresponding to G_i, while G_i^p denotes such total area corresponding to the parent of G_i. If a G_i does not satisfy the inequality (5.14), then we go to the children of G_i and check the children of G_i one by one (still against G_i^p), so on and so forth. Below is the algorithm.

Example 5.14. *In Figure 5.9, suppose that we choose $\rho = 1/3$. G_6 does not follow the inequality in line 5; thus we link the root to G_3 and G_4 (thus, remove G_6). However, G_5 follows the inequality; consequently G_5 is used as the root to call Algorithm 5.7. None of G_1 and G_2 follows the inequality (5.14). Therefore, the final result is that in these three gCaR-trees, the root has three children corresponding to G_5, G_3, and G_4, respectively; the next level contains all sampled instances.*

In our algorithm, those gCaR-trees are pre-computed. We assign 1/4 to ρ since it leads to a very good performance according to our initial experiments.

Calculating-*pscore*^{r}. Our algorithm closely follows the framework of the exact algorithm with the following modifications. Let λ_k denote the smallest *pscore*^{r} value of the current top-k candidates.

1: Pruning Rules.

Algorithm 5.7 gCaR Post-processing

Input: n gCaR-trees; the root Gr of the gobal tree; $0 < \rho \leq 1$.

Output: n gCaR-trees following the inequality (5.14).

Description:

```
1: $\mathcal{Q} := \{$children of $Gr\}$;
2: while $\mathcal{Q} \neq \emptyset$ do
3:   get a $G$ from $\mathcal{Q}$;
4:   $\mathcal{Q} := \mathcal{Q} - \{G\}$;
5:   if $\frac{area(G)}{area(Gr)} < \rho$ then
6:     if $G$ is not children of $Gr$ then
7:       modify the global tree (thus $n$ gCaR-trees) by using $Gr$ as the parent of $G$;
8:     end if
9:     call Algorithm 5.7 with $G$ as the root if $G$ is not a data point;
10:  else
11:    add children of $G$ to $\mathcal{Q}$;
12:  end if
13: end while
```

For each object, we search for the $(q * m)$th greatest homo-dominating number $pscore^r$ among the sampled instances. Below are the pruning rules that we will use in our randomized algorithm.

Pruning Rule 5.4. *$\forall U$, if $pscore^r(U) \leq \lambda_k$, then U can be excluded from the solution of PtopkQ.*

Note that at the object level, we use the number of objects (totally or partially) dominated by U as an upper-bound of $pscore^r(U)$ for applying Theorem 5.4. Let $\mathcal{M}_{\leq\lambda_k,U_i}$ denote the number of sampled instances, from U_i, with their $pscore^r \leq \lambda_k$, respectively.

Pruning Rule 5.5. *An object U_i can be excluded from the solution of PtopkQ (against the probability threshold q) if $\mathcal{M}_{\leq\lambda_k,U_i} \geq (1-q)\times m+1$.*

Note that Pruning rule 5.5 will be used to replace Pruning Rules 5.2 and 5.3 in the exact algorithm.

2: Step 1 and 2 - Pre-ordering and Initial computation

While the Step 1 (pre-ordering objects) in our randomized algorithm is the same as Step 1 in the exact algorithm, Step 2 for computing scores of the first k objects is conducted differently. We need to compute $pscore^r$ for each object instead of $pscore$. Below is the algorithm, Algorithm 5.8, to calculate the $pscore^r$ for one object.

Algorithm 5.8 Calculating $pscore^r$

Input: U; $PD(U)$; $FD(U)$; $0 < q \leq 1$; m samples.

Output: $pscore^r$ of U;

Description:

```
1: for each sampled $u \in U$ do
2:     compute $pscore^r(u)$ against $PD(U)$;
3:     $\delta :=$ the $(q \times m)$th largest value of $pscore^r(u)$;
4: end for
5: $pscore^r(U) := |FD(U)| + \delta$;
```

Note that the computation of $pscore^r(u)$ ($\forall u \in U$) is conducted within the sample that u belongs to. We incrementally maintain a min-heap [CLRS01] against the current top-$(q \times m)$ instances (i.e., with the largest $pscore^r$s) or a max-heap against the current bottom-$[(1-q)m+1]$ objects depending on whether $q \leq 0.5$. Clearly, Algorithm 5.8 runs in time $O(m|PD(U)| + m\log(q * m))$ for each object U.

3: Step 3 - Final Computation.

In this step, we use the same bounding-pruning-refining framework as in the exact algorithm by effectively using the following Theorem 5.7 in combination with Pruning Rules 5.4 and 5.5. Let $\nu_{i,j}$ denote the largest number of instances homo-dominated by an instance contained by an entry $E_{i,j}$ of a gCaR-tree of object U_j. For example, regarding the example in Figure 5.9, $\nu_{6,1} = 2$.

Theorem 5.7. *Suppose that an $E_{i,j}$ fully homo-dominates l_1 entries and partially homo-dominates l_2 entries. Further suppose that $\imath_{E_{i,j}}$ dominates $\imath_{E_{i,j'}}$. Then,*

1. $\nu_{i,j} \leq l_1 + l_2$,
2. $\nu_{i,j'} \leq l_1 + l_2$.

Theorem 5.7 is immediately based on the definitions, and is used in level-by-level computation. Below we present our algorithm details. It also consists of 3 steps: Step 3.1, Step 3.2, and Step 3.3.

Step 3.1: pruning at the object level. It is the same as Step 3.1 in the exact computation (Algorithm 5.4).

Step 3.2: level-by-level pruning. The basic idea is to synchronously traverse the gCaR-trees of a U_j and the uncertain objects in $PD(U_j)$. For an object U_j, let $L^+_\kappa(U_j)$ denote the set of entries at the κ level of the gCaR-tree such that for each entry $E_{i,j}$ in $L^+_\kappa(U_j)$, $\mu_{i,j}$ is not captured less than λ_k. In our algorithm, we initialize each object U_j by assigning 0 to $\mathcal{M}_{\leq\lambda_k,U_j}$ and the root entry of U_j to $L^+_1(U_j)$. The step proceeds as follows for each remaining object U_j.

At each level κ, for every entry $E_{i,j}$ in $L^+_\kappa(U_j)$ we compute l_1 and l_2. If l_1+ $l_2 \geq \lambda_k$, we add the child entries, which are not marked out, of $E_{i,j}$ to $L^+_{\kappa+1}(U_j)$ for the computation at the next level.

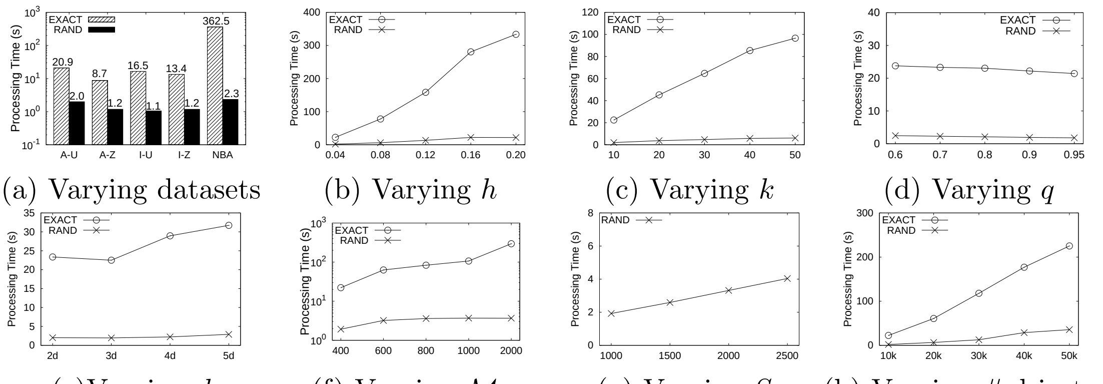
Otherwise ($l_1 + l_2 \leq \lambda_k$), according to Theorem 5.7 we do the following two things.

1. For every entry $E_{i,j'}$ ($\forall U_{j'} \in PD(U_j)$) such that $U_{j'}$ has not been processed in Step 3 and $E_{i,j'}$ is not marked out, if $\imath_{E_{i,j}} \prec \imath_{E_{i,j'}}$, $\mathcal{M}_{\leq \lambda_k, U_{j'}} = \mathcal{M}_{\leq \lambda_k, U_{j'}} + a_{i,j'}$.[6] $U_{j'}$ will be marked out for further consideration if the updated $\mathcal{M}_{\leq \lambda_k, U_j'} \geq (1-q) \times m + 1$ (Pruning Rule 5.5). In case that $U_{j'}$ cannot be excluded, $E_{i,j'}$ is marked out by using Algorithm 5.5; that is, it will not be considered while processing $U_{j'}$.

2. Update $\mathcal{M}_{\leq \lambda_k, U_j}$ to $\mathcal{M}_{\leq \lambda_k, U_j} + a_{i,j}$. Then exclude U_j from the result set if $\mathcal{M}_{\leq \lambda_k, U_j} \geq (1-q) \times m + 1$ (Pruning Rule 5.5).

If U_j is not pruned in Step 3.2, then we invoke Step 3.3.

Step 3.3: final computation. At the leaf level, for all instances in the remaining entries of U_j we compute their actual values of $pscore^r$ and return the $(q \times m)th$ largest value as $pscore^r(U_j)$. If $pscore^r(U_j) > \lambda_k$, then we replace the object with the smallest $psocre^r$ (i.e., λ_k) among the current top-k objects by U_j and update λ_k.



(a) Varying datasets (b) Varying h (c) Varying k (d) Varying q

(e)Varying d (f) Varying $\mathcal{M}$ (g) Varying S (h) Varying #objects

Figure 5.10: Runtime with respect to Different Parameters.

[6]To avoid to over-count already marked-out entries, $a_{i,j'}$ is the number of instances in the subentries of $E_{i,j'}$ that have not been marked out. To efficiently record such $a_{i,j'}$ for each entry, we apply Algorithm 5.5.

5.5 Experimental Study

In this section, we present a thorough performance evaluation of the efficiency and effectiveness of our algorithms. All algorithms are implemented in C++. Experiments are run on PCs with Intel P4 2.8GHz CPU and 2G memory under Debian Linux.

We refer to the exact algorithm in Section 5.3 as **EXACT**, and to the randomized algorithm in Section 5.4 as **RAND**.

Two types of datasets are used in our evaluation process.

Real dataset is extracted from NBA players' game-by-game statistics (http://www.nba.com), containing 339,721 records of 1,313 players. Performance of a player is treated as an uncertain object and the statistics of a player in a single game is treated as an instance of an uncertain object. For one player, all instances are assumed to take the same probability to appear. In our experiment, we use three attributes, points, assistances, and rebounds in an instance. Since larger values of those attributes are preferred, we adopt the corresponding negative values[7].

Synthetic datasets are generated using methodologies in [BKS01] with respect to the following parameters. Dimensionality varies from 2 to 5 with default value **3**. Data domain along each dimension is $[0, 1]$. Number of objects varies from $10,000$ to $50,000$ where default value is $\mathbf{10,000}$. Number of instances per object follows a uniform distribution in $[1, \mathcal{M}]$ where $\mathcal{M}$ changes from 400 to $2,000$ with the default value **400**. Each MBB to bound an uncertain object is a hype-cube; and the average edge length of MBB of uncertain objects follows a normal distribution

[7]Note that there might be correlations among the player statistics. We ignore the correlations so that NBA data can be used to test efficiency and effectiveness of our techniques.

in the range [0,h] with the expectation value $h/2$ and standard deviation 0.025; the default value of h is 0.04 — 4% of the edge length of the whole data space. The value k in PtopkQ varies from 10 to 50 with default value 10. As for randomized algorithm, sample size varies from 1000 to $2,500$. Table 6.2 summarizes parameter ranges and default values (in bold font). Note that in the default setting, the total number of instances is about 2 millions.

Instances of an object follow either *uniform* (random) or *zipf* distribution. In *uniform* distribution, instances are distributed uniformly inside the uncertain range with the same occurrence probability. In *zipf* distribution, firstly an instance u is randomly generated and the distances from all other instances to u follow a zipf distribution with $z = 0.5$. The occurrence probability for each instance also follows zipf distribution with $z = 0.2$.

Centers of objects (objects' MBBs) follow either *anti-correlated* or *independent* distribution. So, in all we have four types of synthetic datasets combining object centers and instances distribution: Anti-Uniform, Inde-Uniform, Anti-Zipf and Inde-Zipf. These are abbreviated to **A-U**, **I-U**, **A-Z**, and **I-Z** in our experiment reports.

dimensionality d	2, **3**, 4, 5
number of objects	**10k**, 20k, 30k, 40k, 50k
edge length h	**0.04**, 0.08, 0.12, 0.16, 0.20
number of instances $\mathcal{M}$	**400**, 600, 800, 1k, 2k
k	**10**, 20, 30, 40, 50
q	0.6, 0.7, 0.8, **0.9**, 0.95
sample size S	**1k**, 1.5k, 2k, 2.5k
data types	**A-U**, A-Z, I-U, I-Z, NBA

Table 5.4: Parameter Values.

5.5.1 Efficiency Evaluation

We evaluate our algorithms against the parameters in Table 6.2.

Overall Performance. Figure 5.10(a) reports the result of our performance evaluation over synthetic (with the default setting) and real datasets. The experiment demonstrates that while EXACT is very efficient against various synthetic datasets with the default setting, it is slower against the NBA dataset. This is because in the NBA dataset, MBB sizes are large relative to the whole data space; this gives a very high overlapping degree among objects' MBBs. On the other hand, RAND very effectively deals with such situation. RAND has a very steady performance and is at least 10 times faster than EXACT against all these datasets. We run the trivial exact algorithm as discussed in Section 5.1.3; that is, compute *pscore* for each object and then choose the top-k. Our experiment results show that it is about 100 times slower than EXACT. We also implement the trivial randomized algorithm as discussed in Section 5.4; that is, compute $pscore^r$ for each instance in a sample. The costs are 1589(s), 1543(s), 3081(s), 3376(s), and 115(s), respectively; it increases to 6685(s) when 2500 samples are used. Consequently we omit the evaluation of both trivial algorithms in the rest of our experiments. Note that the trivial randomized algorithm runs fast against NBA data; this is because NBA data only have about 1000 objects.

Varying MBB sizes, k and q. Figure 5.10(b) reports our second experiment results, against synthetic datasets with different average MBB sizes. Figure 5.10(c) reports our performance evaluation against different k values. While the costs of EXACT linearly increase when k increases, the performance of RAND is quite steady. This is because that the costs of computing the scores, $pscore^r$, for objects in RAND are no longer as dominant as that in EXACT. The experiment results, depicted in Figure 5.10(d), show the impact from different q values is quite minor.

Varying other parameters. Figures 5.10(e) - (g) report the possible impacts against dimensionality, average instance numbers, and average sample size. It is interesting to note that the costs of EXACT generally increase with the increment of dimensionality but the costs in $3d$ are slightly less than that in $2d$; this is because the ratio of average MBB volume against the data space decreases with the increment of dimensionality. Nevertheless, the experiment demonstrates that an increment of dimensionality plays a dominant role in the costs from $3d$.

The impact of the number of objects is plotted in Figure 5.10(h). Although the processing time of two algorithms both increases as more uncertain objects are involved, RAND has overall better performance and also degrades much more slowly than EXACT.

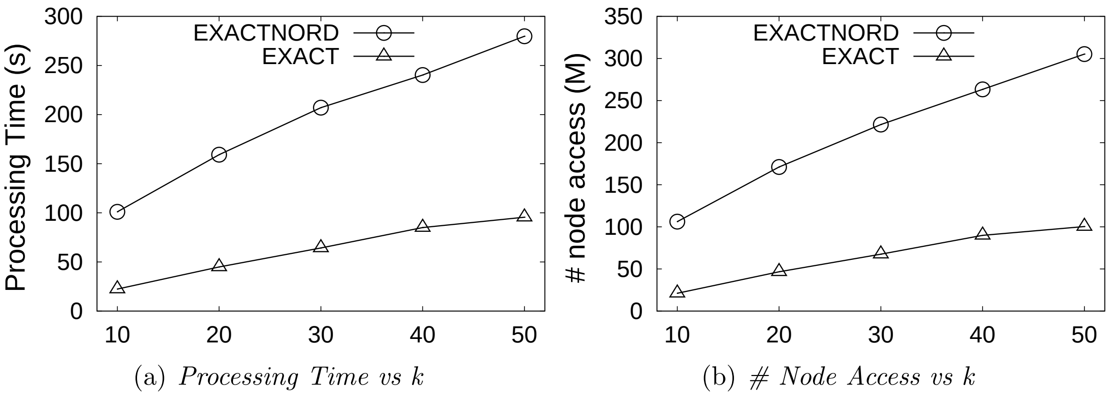


Figure 5.11: Performance vs Diff. Object Access Orders

Accessing order. In order to evaluate the effectiveness of the objects accessing order in Section 5.3.1, we also implement another version of the exact algorithm, named EXACTNORD, in which the objects are accessed with a random order. We evaluate the processing time as well as the number of node access of two algorithms with k varying from 10 to 50 in Figure 5.11. As depicted in Figure 5.11(a), the accessing order plays an important role for the computation as the EXACT algorithm significantly outperforms the EXACTNORD. We also use the *warm-buffer* paradigm to run our algorithms to evaluate I/O costs. In Figure 5.11(b), we record the number of node access for the aR-Trees of the uncertain objects during the computation. As expected, the number of node access of EXACT Algorithms is much less than that of EXACTNORD.

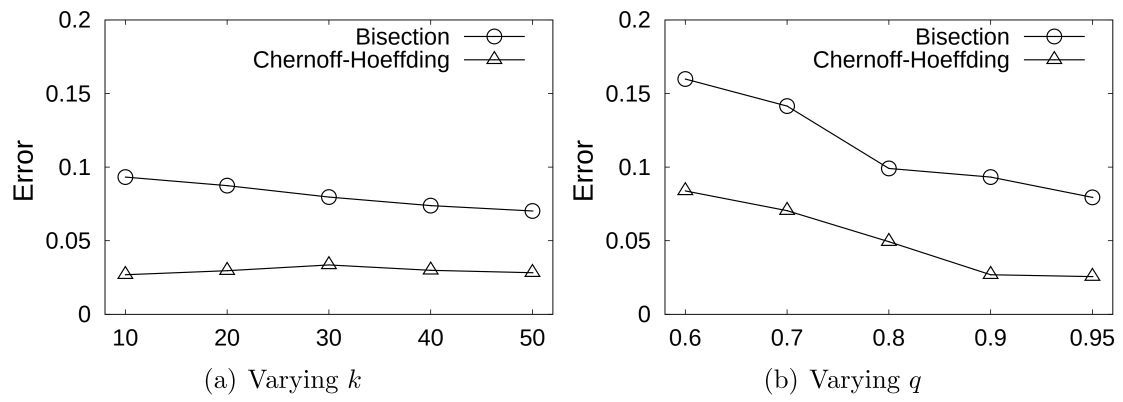


Figure 5.12: Chernoff-Hoeffding based vs Bisection based

5.5.2 Pruning Powers

Chernoff-Hoeffding vs Bisection. We first evaluate the effectiveness of the Chernoff-Hoeffding-bound based upper bound and the Bisection-based upper bound. The experiment is conducted against the real data - NBA dataset. In our experiment, we first vary k values and then vary q values. We record the average value of

$$P^{upper}_{\geq\lambda_k}(U) - P_{\geq\lambda_k}(U)$$

during query processing where $P_{\geq\lambda_k}(U)$ is the actual probability and $P^{upper}_{\geq\lambda_k}(U)$ represents the Chernoff-

Hoeffding-bound based upper bound and the Bisection-based upper bound, respectively. Note that for a fair comparison, we only record such average for the

Bisection-based upper bounds when Chernoff-Hoeffding Bound based upper bound can be used. The results are reported in Figure 5.12(a) and Figure 5.12(b). They demonstrate that the Chernoff-Hoeffding Bound based upper bound is tighter than the Bisection-based upper bound. This is the reason that in our algorithm, we employ the Chernoff-Hoeffding Bound based upper bound whenever applicable.

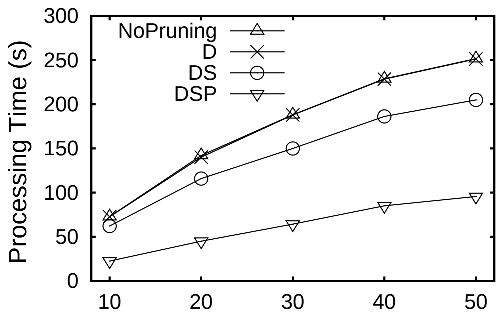


Figure 5.13: Varying k

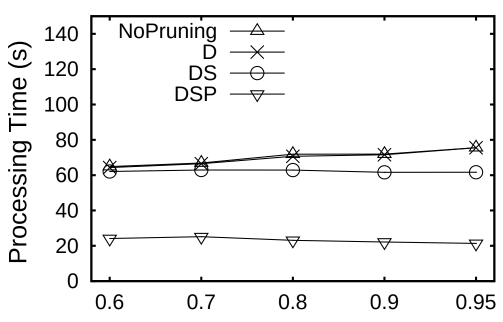
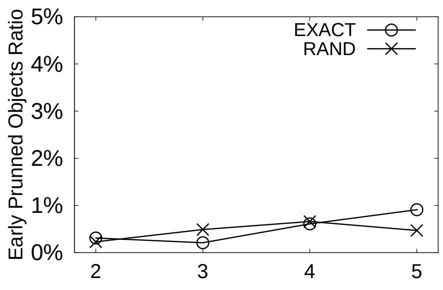
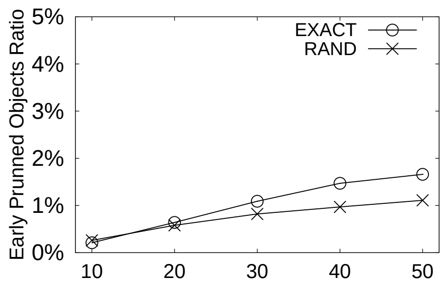


Figure 5.14: Varying q

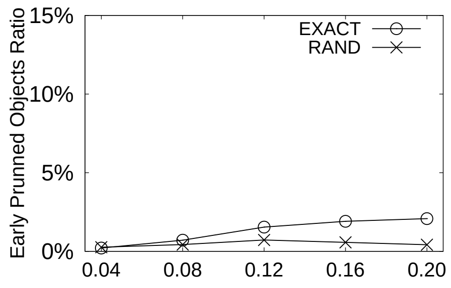
Various Pruning Techniques. Figures 5.13 and 5.14 report our evaluation of the effectiveness of the pruning rules presented in the chapter with various k values and q values, respectively. NoPruning denotes the exact algorithm without applying any pruning rules in Section 5.3.3 at the instance levels, D denotes that we apply the *Drill-down*-based pruning rule, DS denotes that we apply the *Drill-down*-based pruning rule and the *Score* based pruning rule at each level, and DSP denotes that we apply *Level's Probability*-based pruning rule (i.e. Chernoff-Hoeffding Bound based upper-bound and the Bisection-based upper-bound) each level in addition to DS. They demonstrate that an application of the *Drill-down*-based pruning rule alone does not improve much efficiency since it basically still functions at the object level. While combining with level-by-level score based pruning rule does improve efficiency noticeably, adding Chernoff-Hoeffding Bound based upper-bound and the Bisection-based upper-bound significantly improves the performance. This is because the computation costs at each level are significantly reduced by using those



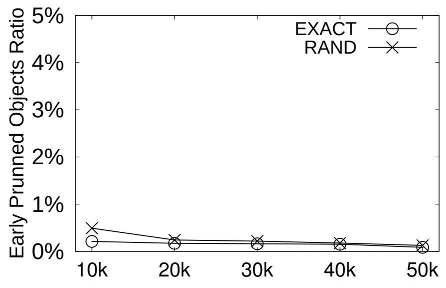
(a) Varying d



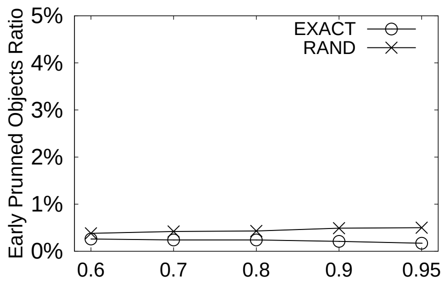
(b) Varying k



(c) Varying h



(d) Varying n

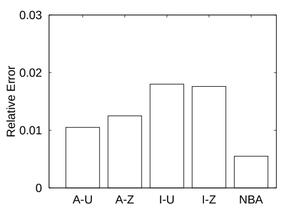


(e) Varying q

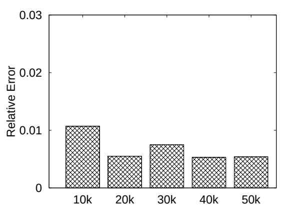
Figure 5.15: Node Calculated Ratio with respect to Different Parameters.

upper-bounds and the upper-bounds are tight. Note that NoPruning is basically the combination of the techniques in [YM07] and techniques in [HPZL08b, YLKS08] on the top of our pre-ordering techniques.

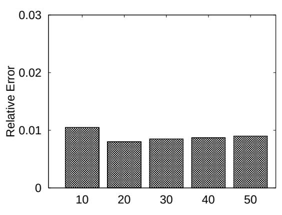
Effectiveness. We report our performance evaluation of pruning power of EXACT and RAND in Figure 5.15 against dimensionality, k values, edge lengths, object numbers, and q values. The experiment is conducted against syntectic data in order to evaluate all possible impacts. We record “early pruned object ratio” - the ratio of the number of objects, with entries of local aR-trees accessed from the 2nd



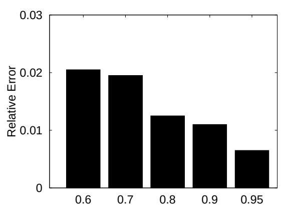
(a) Varying *dataset*



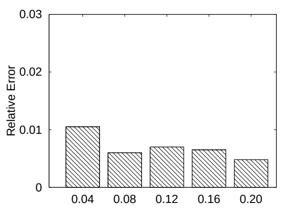
(b) Varying n



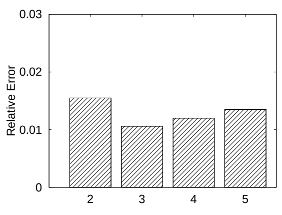
(c) Varying k



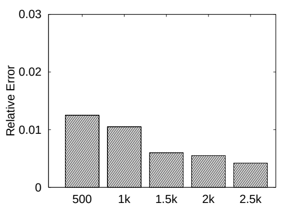
(d) Varying q



(e) Varying h



(f) Varying d



(g) Varying S

Figure 5.16: Relative error with respect to Different Parameters

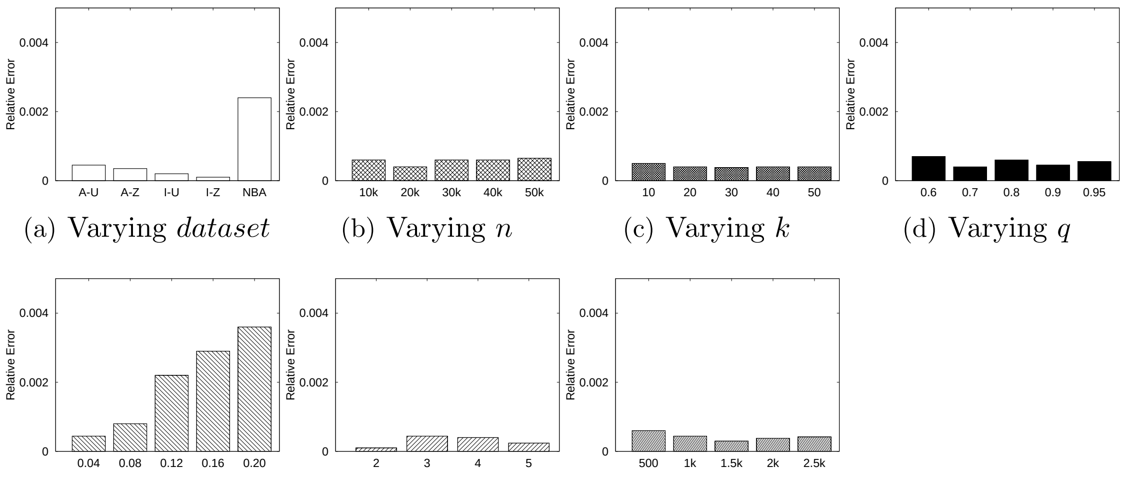
level onwards, over the total number of objects. Our evaluation reports that the exact algorithm has a very powerful set of pruning techniques and up to 97% of objects have been pruned from the candidate sets even when MBB is large.

5.5.3 Accuracy Evaluation

We evaluate possible impacts of different parameters on the accuracy of RAND. Evaluation is based on average *relative errors* for a retrieved object's dominating probability with its $pscore^r$ computed using RAND regarding a given threshold q. We use the following relative error metrics to evaluate the ability of RAND to meet a given threshold q. Without loss of generality, U_i denotes the top-ith object returned by RAND.

$$err_i^p = \begin{cases} 0 & \text{if } P_{\geq pscore^r(U_i)} \geq q \\ \frac{|P_{\geq pscore^r}(U_i) - q|}{q} & \text{otherwise.} \end{cases}$$

Figure 5.16 reports our performance evaluation, regarding the average relative error ($\frac{\sum_{i=1}^{k} err_i^p}{k}$), against data types, number of objects, k values, q values, different average MBB sizes, dimensionality, and sample sizes. Our experiment results



(a) Varying *dataset* (b) Varying n (c) Varying k (d) Varying q

(e) Varying h (f) Varying d (g) Varying S

Figure 5.17: Relative error of score with respect to Different Parameters

demonstrate that when sample size reaches 1000, the relative error is already very small. Moreover, the accuracy is not quite related to the dimensionality, object number, k values, or MBB sizes. Nevertheless, the accuracy decreases when q gets smaller; this is because when q is smaller, RAND requires more samples to retain the same accuracy according to our theoretic results in Section 5.4. It also shows that the accuracy increases when the sample size increases.

We also evaluate the accuracy in the top-k scores output by RAND using the average relative error metrics - $\frac{\sum_{i=1}^{k} err_i^l}{k}$ where

$$err_i^l = \begin{cases} 0 & \text{if } P_{\geq pscore^r}(U_i) \geq pscore_i \\ \frac{|pscore^r(U_i) - pscore_i|}{pscore_i} & \text{otherwise.} \end{cases}$$

As demonstrated in Figure 5.17, the performance of RAND is very accurate - the average relative error is less than 0.4%. It is interesting to note that such accuracy is not quite related to these parameters.

5.5.4 Summary

Both of EXACT and $RAND$ are efficient when k is not very large (a typical case for a top-k query), the average MBB size of uncertain objects is reasonable (say, upto 20% of the edge length of the data space), and the total data size is about a few millions. Nevertheless, our randomized algorithm is much more efficient and is also very scalable against dimensionality, k values, data sizes, and object MBB sizes; it is also highly accurate when the sample size reaches 1000.

5.6 Conclusion

In this chapter, we formally define a probabilistic threshold top-k dominating query. To process such a query, we firstly propose an exact algorithm. The exact algorithm utilizes novel and efficient pruning techniques based on novel mathematic characterizations. While fairly efficient, it is quite sensitive to sizes of data set, uncertain object sizes, k values, etc. To trade-off between efficiency and accuracy, a randomized algorithm with an accuracy guarantee is proposed together with a new data structure, gCaR-tree; it is much more efficient than the exact algorithm. The efficiency and effectiveness of these two algorithms are extensively investigated in experimental study.

Note that our algorithms are main memory based. It can be immediately extended to external memory computation using warm buffer; that is, keep things in the buffer and use a buffer replacement policy once it is full. We have evaluated the I/O costs for such a paradigm. Moreover, our techniques developed in the chapter can be immediately extended to cover the dual problem. That is, given a threshold about the number of objects to be dominated, find top-k objects with the maximum dominating probabilities. Finally, our randomized algorithm can also be immediately extended to continuous cases by sampling PDFs using Monte Carlo sampling [KW86].

Chapter 6

Quantile-Based KNN Over Multi-Valued Objects [1]

Given a set $\mathcal{D}$ of objects (points) in a d-dimensional metric space and a d-dimensional query object (point) q, the K nearest neighbor search retrieves the K closest objects to q from $\mathcal{D}$. The conventional KNN search has been extensively studied [HS99, RKF95] with a wide spectrum of applications including data mining, contents-based image retrieval, and location based services. In this chapter, we study the problem of K nearest neighbor search over objects each of which has a collection of values (instances) without temporal constraints specified; that is, we do not deal with sequence databases [AFS93, KS95].

The existing model, probabilistic KNN, is to apply the uncertain semantics to each object by treating the collection of instances of each object mutually exclusive. It aims to catch relative distributions among objects with multi-instances.

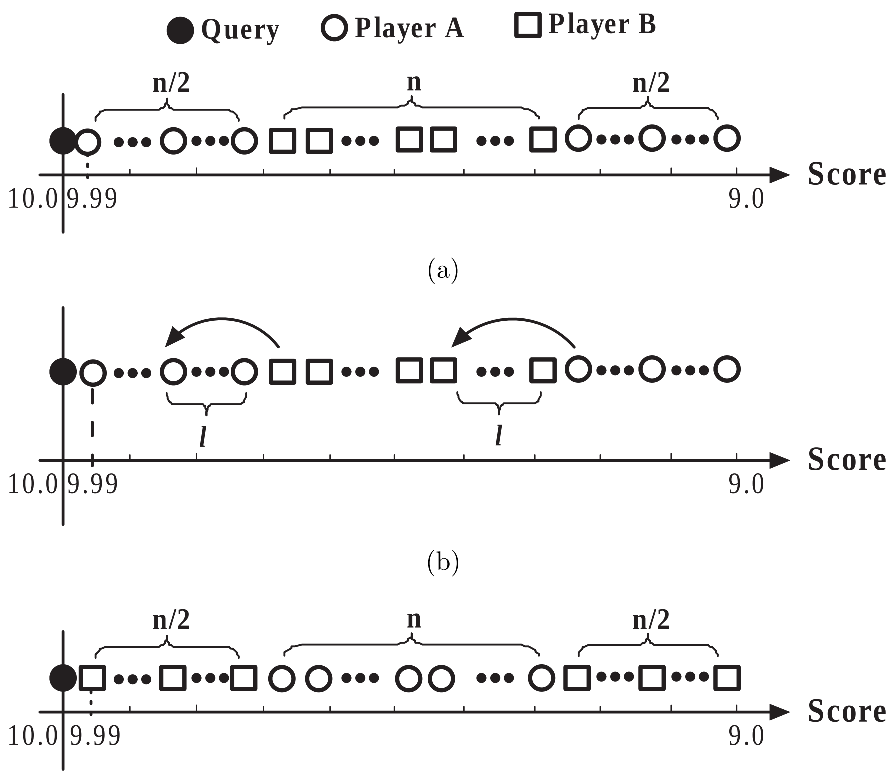
[1]The techniques presented in this chapter originally appear in the paper "Quantile-Based KNN Over Multi-Valued Objects", Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, Ying Zhang and Wei Wang, *to appear in 26th International Conference on Data Engineering (ICDE), 2010*

The two semantics of ranking top-k uncertain tuples are employed in a probabilistic KNN model: 1) retrieving k tuples that can co-exist in a possible world (e.g. U-top$k$) [SIC07], and 2) retrieving tuples according to the probability that a tuple is top-k or at a specific rank in all possible worlds (e.g. U-kRanks and PT-k) [SIC07, HPZL08b] [2]. While various probabilistic NN models are proposed in [BSI08, CCMC08, KKR07], a probabilistic KNN model over uncertain data has been proposed following U-top$k$ ranking semantics [CCCX09]. In these probabilistic KNN models, the probability for an object to be KNN to a query object is calculated to define the result of a KNN. Nevertheless, below we show that the probabilistic KNN models may provide results *insensitive* to relative distributions of instances of objects.

Motivating Example. Let $k = 1$. In gymnastics, suppose that we want to select the "best" balance-beam player among all candidates to participate a world championship. The scores of two players A and B, based on the most recent n games/attempts, are depicted in Figure 6.1(a), respectively. Assume that the $2n$ scores of A and B (n for A and n for B) are distributed from 9.99 to 9.0 as depicted in Figure 6.1(a).

Assume that we approximately treat each player as an uncertain object and the score of an attempt as an instance with the equal occurrence probability. It can be immediately verified that based on the existing probabilistic NN models, player A and player B have the same probability, $\frac{1}{2}$, to be the nearest neighbor of the query point q (i.e. the score 10) if $|10 - score|$ is used as the distance metric. We permutate the distribution in Figure 6.1(a) by swapping the two pairs of instances of A and B as depicted in Figure 6.1(b). It is immediate that A and B still have the same probability, $\frac{1}{2}$, to be the nearest neighbor regarding the score

[2]When $k = 1$, these two models are the same.



(a)

(b)

(c)

Figure 6.1: Motivating Example

distribution after these two permutations. Choosing $l = \frac{n}{2}$, the score distribution in Figure 6.1(a) is eventually modified to the score distribution in 6.1(c) after $\frac{n}{2}$ such pairs of permutations; consequently, the nearest neighbor probabilities of A and B, respectively, remain unchanged, $\frac{1}{2}$, regarding the distribution in Figure 6.1(c).

Quantile-Based KNN. The examples in Figures 6.1 (a)-(c) demonstrate that the existing probabilistic KNN models may be insensitive to relative distributions of object instances. Very recently, in [CLY09] a novel model based on the *expected* rank for ranking top-k uncertain objects has been proposed. Regarding the distributions and the permuted intermediate distributions as depicted above in Figures 6.1 (a)-(c), player A and B always have the same expected rank. Moreover, in the above application we do not need to enforce the uncertain semantics among multi-instances of each player by treating them mutually exclusive. Motivated by these, we treat each player as a multi-valued object.

Quantiles [YMT06] may provide a succinct summary of data distributions. In this chapter, we investigate the KNN problem over multi-valued objects based on a ϕ-quantile distance ($\phi \in (0, 1]$) from a multi-valued object to a query Q; for example, the median is the 0.5-quantile. We extend our investigation to the KNN problem over multi-valued objects based on overall distances in the "best population" (with a given size specified by ϕ-quantile) regarding each object; such overall distances are called a ϕ-quantile group-base distance.

Regarding the above example, our KNN problem based on 0.5-quantile distances is to rank players based on their median performances, respectively. The KNN problem based on a 0.5-quantile group-base distance is to rank players based on their overall performances of the top-50% of scores, respectively.

The above example contains multi-valued objects in a 1-dimensional space and the query is a single-valued point. Nevertheless, our investigation covers the applications where data objects consist of multiple instances in a d-dimensional space and a query object may also consist of multiple instances in a d-dimensional space. For instance, in NBA the performance of a player per game may be measured by his statistics (scores, assists, rebounds, steals, blocks) and may be treated as an instance of the player; consequently, each player has a set of instances. Suppose that a team wants to sign a contract with player A and wants to find his market value. The team may want to find out the top-k "similar" NBA players, with existing contracts, to A against their recent game statistics. Then, the team can use the salaries information of these k-players to project the salary level of A.

Contributions. To the best of our knowledge, this is the first work to study KNN problems regarding quantiles over multi-valued objects. Yiu *et al* [YMT06] develop efficient techniques to compute quantile-distances among data points; nevertheless the techniques are not applicable to our problem due to the following reasons. Firstly, the query object in our problem setting may have multiple instances and we count all pair combinations between an object and a given query object, while the computation of multi-source-based quantile-distances in [YMT06] is to compute the distance of an instance to its nearest given source. Secondly, the quantile group-base distance problem studied in this chapter is NP-hard. Our contribution may be summarized as follows.

- We make the first attempt to identify KNN sensitive to the relative distributions among multi-valued objects.
- Efficient, novel techniques are proposed for computing quantile distance based KNN against a set of multi-valued objects and a given query object that is also multi-valued.
- We show that the problem of KNN against the quantile group-base distance is NP-hard. Novel and efficient algorithms are proposed with the approximation ratio 2.

As a byproduct, our techniques to compute a ϕ-quantile distance is $O(n)$ if single-valued object is involved while the technique in [YMT06] is $O(n \log n)$ where n is the number of instances. Besides the theoretical analysis, an extensive performance evaluation demonstrates that the proposed techniques are both efficient and effective.

The rest of the chapter is organized as follows. In Section 6.1, we formally define the problems and provide some necessary background information. In Section 6.2, we present the framework of our algorithms to conduct KNN against these 2 quantile-based KNN problems. Section 6.3 and Section 6.4 present query processing techniques for these two KNN problems, respectively. In Section 6.5, we report our experiment results. This is followed by conclusions.

6.1 Background Information

We present problem definition and necessary preliminaries. For reference, notions frequently used in the chapter are summarized in Table 6.1.

Notation	**Definition**
$\mathcal{U}$	set of of objects
U (Q)	multi-valued (query) object
E	entry of R-tree
u (q)	instance of U (Q) - a point in d-dimensional space
$w(u)$ $(w(S))$	(total) weight of u (the set S)
$d(q,u)$	Euclidean distance between q and u
$d^{lo}(E,E')$	distance lower-bound between E and E'
$d^{up}(E,E')$	distance upper-bound between E and E'
$d_\phi(Q,U)$	ϕ-quantile distance of Q and U
$gbd_\phi(Q,U)$	ϕ-quantile group-base distance of Q and U
$Q \times U$	Cartesian product of instances from Q to U

Table 6.1: The Summary of Notations.

6.1.1 Problem Definition

Given a collection S of m elements, each element s_i has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) = 1$. Let S be sorted increasingly on a search key f - a function; that is, $f(s_i) \leq f(s_j)$ if $i < j$. Without loss of generality, in this chapter a *sorted* collection S of data elements, thereafter, always means sorting S *increasingly* on a given search key unless otherwise specified.

Definition 6.1 (ϕ-quantile of S)**.** *Given a ϕ ($0 < \phi \leq 1$), the ϕ-quantile S_ϕ of S is the* **first** *element s_i in the* **sorted** *S on the search key such that $\sum_{j=1}^{i} w(s_j) \geq \phi$.*

In our problem definition, an instance of an object U (or Q) is weighted - weight gives the *representativeness* of an instance in U. For instance, in the motivating examples in introduction, a game statistic may appear multiple times; consequently a normalized weight (the occurrence of an instance over the total occurrences of all instances) may be used to indicate the representativeness of an instance. Note that the total of such weights in U (or Q) is 1.

A multi-valued object U is represented as $\{(u_i, w(u_i))|1 \leq i \leq m\}$ where u_i is a point in a d-dimensional space, $0 < w(u_i) \leq 1$ $(1 \leq i \leq m)$, and $\sum_{i=1}^{m} w(u_i) = 1$. A query object Q is also a multi-valued object. We use $\mathcal{U}$ to denote a set of multi-valued objects.

For a given Q and each $U \in \mathcal{U}$, there are totally $(|Q| \times |U|)$ pairs of instances in $Q \times U$ where each pair (q_i, u_j) ($q_i \in Q$ and $u_j \in U$) has the weight $w(q_i) \times w(u_j)$, namely $w(q_i, u_j)$. Clearly, $\sum_{q_i \in Q, u_j \in U} w(q_i) \times w(u_j) = 1$. The *Euclidean distance* $d(q_i, u_j)$[3] between q_i and u_j is called the distance of (q_i, u_j). Let $Q \times U = \{((q_i, u_j), w(q_i, u_j)) \mid q_i \in Q \;\&\; u_j \in U\}$.

Definition 6.2 (ϕ-quantile distance of Q and U)**.** *Given a $\phi \in (0, 1]$, let $Q \times U$ be sorted increasingly on the search key - the distance $d(q_i, u_j)$ of each element (q_i, u_j). Then, the distance of the ϕ-quantile of $Q \times U$ is called the* ϕ-quantile distance *of $Q \times U$, denoted by $d_\phi(Q, U)$.*

Definition 6.2 states that if (q, u) is the ϕ-quantile of $Q \times U$ (i.e., $(Q \times U)_\phi = (q, u)$) then $d(q, u)$ is $d_\phi(Q, U)$.

Example 6.1. *Regarding the example in Figure 6.2, $|Q| = 3$ and $|U| = 2$. Assume that $w(q_1) = \frac{1}{2}$, $w(q_2) = w(q_3) = \frac{1}{4}$; $w(u_1) = w(u_2) = \frac{1}{2}$. Consequently, $Q \times U$ consists of the following six pairs sorted on their distances increasingly:*

[3]Note that our techniques developed in this chapter is based on Euclidean distance; nevertheless they can be immediately extended to cover other distance metrics.

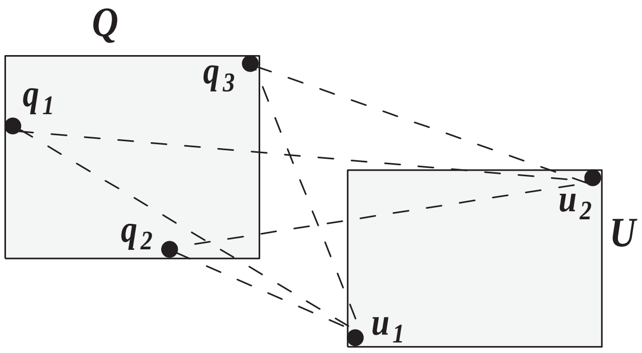


Figure 6.2: Distances between 2 Multi-Valued Objects

$Q \times U = \{((q_2,u_1),\frac{1}{8}),\ ((q_3,u_1),\frac{1}{8}),\ ((q_3,u_2),\frac{1}{8}),\ ((q_1,u_1),\frac{1}{4}),\ ((q_2,u_2),\frac{1}{8}),\ ((q_1,u_2),\frac{1}{4})\}$.

Note that the $\frac{2.5}{8}$*-quantile and the* $\frac{3}{8}$*-quantile of* $Q \times U$ *are the same* (q_3,u_2)*. The* 0.2*-quantile distance* $d_{0.2}(Q,U)$ *of* Q *and* U *is* $d(q_3,u_1)$*,* $d_{0.5}(Q,U)$ *is* $d(q_1,u_1)$*,* $d_{0.6}(Q,U)$ *is also* $d(q_1,u_1)$*.* $\square$

Below, we specify a measure based on aggregates to define the *top/best* quantile-population of S.

Definition 6.3 (ϕ-quantile population of S)**.** *Given a* S *and a* $\phi \in (0,1]$*, a* ϕ*-quantile population* $S_{\phi,P}$ *of* S *is a sub-collection* S' *of* S *such that the total weights of the elements in* S' *is not smaller than* ϕ *and removing any element from* S' *makes the total weights in the remaining sub-collection smaller than* ϕ*.*

Definition 6.4 (ϕ-quantile group-base distance)**.** *Given a* $\phi \in (0,1]$*, the* ϕ*-quantile* group-base *distance of* Q *and* U *is the minimum total weighted distance among* ϕ*-quantile populations of* $Q \times U$*; that is, the minimum value of* $\sum_{(q,u)\in S'} w(q)w(u)d(q,u)$ *with the constraint that* S' *is a* ϕ*-quantile population of* $Q \times U$*.*

The ϕ-quantile *group-base* distance between Q and U is denoted by $gbd_\phi(Q,U)$. Note that for a $\phi \in (0,1]$, the example below shows that $gbd_\phi(Q,U)$ is not always defined on the set of the "consecutive" smallest distances. In fact, we will show in Section 5 that the computation of $gbd_\phi(Q, U)$ is NP-hard.

Example 6.2. *Regarding Example 6.1, let $\phi = 0.5$. $gbd_{0.5}(Q, U) = \frac{1}{8}d(q_2, u_1) + \frac{1}{8}d(q_3, u_1) + \frac{1}{8}d(q_3, u_2) + \frac{1}{8}d(q_2, u_2)$ instead of $\frac{1}{8}d(q_2, u_1) + \frac{1}{8}d(q_3, u_1) + \frac{1}{8}d(q_3, u_2) + \frac{1}{4}d(q_1, u_1)$. Here, $\{((q_2, u_1), \frac{1}{8}), ((q_3, u_1), \frac{1}{8}), ((q_3, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$ is not even a 0.5-quantile population of $Q \times U$.*

In fact, there are several 0.5-quantile populations of $Q \times U$, including $\{((q_3, u_1), \frac{1}{8}), ((q_2, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$, $\{((q_2, u_1), \frac{1}{8}), ((q_2, u_2), \frac{1}{8}), ((q_1, u_1), \frac{1}{4})\}$, etc. □

Definition 6.5 (ϕ-Quantile KNN). *Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a d-dimensional space, and a multi-valued query object Q, the ϕ-quantile KNN problem is to retrieve the set Φ_K of K objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $d_\phi(Q, U) \leq d_\phi(Q, U')$.*

Definition 6.6 (ϕ-Quantile Group-base KNN). *Given a $\phi \in (0, 1]$, a set $\mathcal{U}$ of multi-valued objects in a d-dimensional space, and a multi-valued query object Q, the ϕ-quantile group-base KNN problem is to retrieve the set Φ_K of K objects from $\mathcal{U}$ such that for each $U \in \Phi_K$ and each $U' \in \mathcal{U} - \Phi_K$, $gbd_\phi(Q, U) \leq gbd_\phi(Q, U')$.*

Problem Statement. In this chapter, for a given $\phi \in (0, 1]$, we study the problems of efficiently computing the ϕ-Quantile KNN and the ϕ-Quantile Group-base KNN.

6.1.2 Preliminaries

Given a collection S of m elements, each element s_i has a weight $w(s_i)$ where $0 < w(s_i) \leq 1$ and $\sum_{i=1}^{m} w(s_i) \leq 1$. A naive way to compute the ϕ-quantile is to firstly sort S regarding a given search key f, and then scan the sorted list to obtain the ϕ-quantile of S. Clearly, the naive algorithm runs in $O(m \log m)$.

In [CLRS01], an efficient and effective partitioning technique PARTITIONING (S) is proposed to find an element $s \in S$ to divide S into two sub-collections S_1 and S_2 with the following properties:

1. for each $s' \in S_1$, $f(s') \leq f(s)$; and for each $s' \in S_2$, $f(s') \geq f(s)$.
2. $|S_1| \geq \frac{3}{10}m - 6$ and $|S_2| \geq \frac{3}{10}m - 6$.

Using the partitioning technique, in Algorithm 6.1 we present an iteration-based algorithm to compute a ϕ-quantile when S is not sorted.

Algorithm 6.1 QUANTILE (S, ϕ)

```
Input: $S$: a collection of $m$ elements; $\phi$: $0 < \phi \leq \sum_{i=1}^{m} w(s_i)$; $f$: specify a search
    key;
Output: $\phi$-quantile of $S$
1: ($s$, $S_1$, $S_2$) $\longleftarrow$ PARTITIONING (S);
2: if $\phi \leq w(S_1)$ then
3:     call QUANTILE ($S_1$, $\phi$);
4: else
5:     if $\phi > w(S_1) + w(s)$ then
6:         call QUANTILE ($S_2$, $\phi - w(S_1) - w(s)$);
7:     else
8:         return $s$;
```

In Algorithm 6.1, $w(S_1)$ denotes the total weights of the elements in S_1. When S has only one element, $S_1 = S_2 = \emptyset$.

It is shown in [CLRS01] that the time complexity of PARTITIONING (S) is linear - $O(|S|)$. Consequently, each iteration runs in linear time regarding the current sub-collection size. Recall the property 2 above in PARTITIONING (S). It is immediate that the sizes of sub-collections involved in the iterations in Algorithm

6.1 are exponentially reduced - at the ith iteration bounded by $((\frac{7}{10})^{i-1}m+c)$ where c is a constant; consequently, the time complexity of Algorithm 6.1 is **linear** - $O(m)$. The correctness of Algorithm 6.1 immediately follows from the property 1 of PARTITIONING (S).

6.2 Framework Overview

Our techniques for solving the ϕ-quantile KNN and and the ϕ-quantile group-base KNN for a given $\phi \in (0, 1]$ follow a standard seeding-refinement paradigm outlined in Algorithm 6.2.

Algorithm 6.2 Framework

- **Phase 1 - Seeding:** Compute the ϕ-quantile (or ϕ-quantile group-base) distance from each of the K chosen objects to Q.
- **Phase 2 - Refinement:** Determine the final solution for ϕ-quantile KNN (or ϕ-quantile group-base KNN).

In the seeding phase, we choose K objects and compute their ϕ-quantile distances (or ϕ-quantile group-base distances); assume that γ_k (λ_K) is the maximal of these K ϕ-quantile distances (or the ϕ-quantile group-base distances). In the refinement phase, we use γ_K (λ_K) and ϕ to effectively prune objects and iteratively update γ_K (λ_K) (if necessary).

Selecting K Objects in the Seeding Phase. Any K multi-valued objects from $\mathcal{U}$ could be used for the seeding phase. Clearly, the smaller γ_K is, the more powerful γ_K may be used in the refinement for pruning. In our algorithm, we use the mean a_U of the multiple instances for each multi-valued object U, and we use the mean a_Q of the multiple instances of Q. Then we apply the KNN algorithm in [HS95]

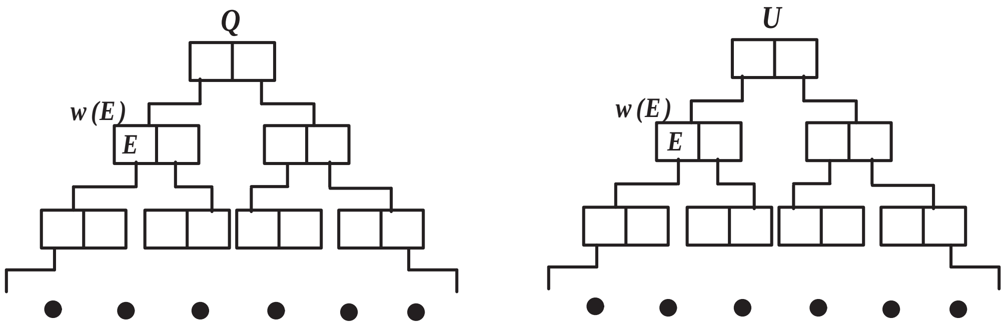


Figure 6.3: Local aR-trees for Multi-Valued Objects

to obtain the K nearest neighbors to a_Q from $\{a_U \mid U \in \mathcal{U}\}$. Subsequently, we use the K objects corresponding to these K nearest means to a_Q as the K objects in the seeding phase for both ϕ-quantile KNN and ϕ-quantile group-base KNN, respectively.

Data Structures. Below are the data structures used in the seeding and refinement phases in our techniques. For each multi-valued object $U \in \mathcal{U}$, a *local* aR-tree is built to organize its multiple instances. The aggregate information kept on each intermediate entry is the sum of weights of instances indexed by the entry. Namely, for every intermediate entry E in the local aR-tree, we record the weight of E as the sum of weights (total weights) of instances having E as an ancestor. Local aR-trees will facilitate the efficient computation of ϕ-quantile (or ϕ-quantile group-base) KNN and support effective pruning techniques. Figure 6.3 shows the local aR-trees for Q and U where each intermediate entry records $w(E)$.

Besides, we maintain an R-tree on the MBBs of multiple instances of objects. That is, for each object we first obtain the MBB of its multiple instances. Then we build an R-tree on these MBBs. This R-tree is called the *global* R-tree. Note in the global R-tree, each leaf is an MBB of an object.

Distances between MBBs. Given two MBBs E and E', we compute the minimal and maximal distances $d^{lo}(E, E')$ and $d^{up}(E, E')$ between them as follows. For each dimension i ($1 \leq i \leq d$), let $I_{E,i}$ and $I_{E',i}$ denote the intervals on which E and E' are projected, respectively. The minimum distance $mindist_i(E, E')$ between $I_{E,i}$ and $I_{E',i}$ is defined as follows. If $I_{E,i}$ overlaps with $I_{E',i}$ then $mindist_i(E, E') = 0$ otherwise $mindist_i(E, E')$ is the minimal value among the distances of 4 pairs of the ends of $I_{E,i}$ and $I_{E',i}$. $maxdist_i(E, E')$ is the the maximal value among the distances of 4 pairs of the ends of $I_{E,i}$ and $I_{E',i}$.

$$d^{lo}(E, E') = \sqrt{\sum_{i=1}^{d}(mindist_i(E, E'))^2}$$
$$d^{up}(E, E') = \sqrt{\sum_{i=1}^{d}(maxdist_i(E, E'))^2}$$

Figure 6.4 below shows representative examples. Note that we can immediately verify that for each pair of instances (u, u') where u is contained by E and u' is contained by E', $d^{lo}(E, E') \leq d(u, u') \leq d^{up}(E, E')$. Thus, $d^{lo}(E, E')$ and $d^{up}(E, E')$ can be used as a lower- and upper- bound of the distance of any pair in $E \times E'$. Immediately, computing minimal/maximal distance between the two d-dimensional MBBs requires $O(d)$ time.

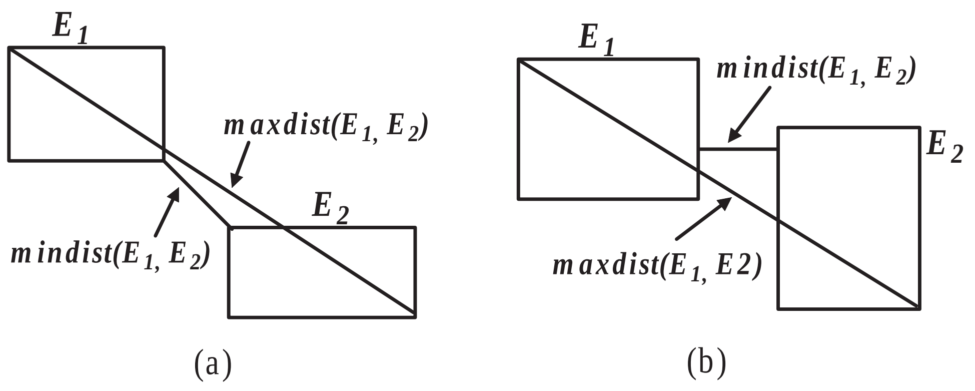


Figure 6.4: Minimal/Maximal Distance between 2 MBBs

6.3 ϕ-Quantile KNN

We present our techniques for conducting ϕ-quantile KNN for a given $\phi \in (0, 1]$. We first present an efficient algorithm to compute a ϕ-quantile distance between Q and U instead of a brute-force computation; this will be used in the two phases. Then, we present a set of novel pruning techniques in the refinement phase, as well as the refinement algorithm.

6.3.1 Efficiently Computing ϕ-Quantile Distances.

Given Q, U, and a $\phi \in (0, 1]$, we present an efficient algorithm to compute $d_\phi(Q, U)$ in this section.

Naive Linear Algorithm. Firstly, for each (q_i, u_j) in $Q \times U$, we calculate its distance $d(q_i, u_j)$ and its weight $w(q_i, u_j)$ $(= w(q_i)w(u_j))$. Then call Algorithm 6.1 to produce the ϕ-quantile (q, u) of $Q \times U$ regarding the search key value (distance) of each (q_i, u_j) and the weight $w(q_i, u_j)$ of each (q_i, u_j). Clearly, $d_\phi(Q, U) = d(q, u)$ and the naive algorithm runs in linear time regarding $|Q \times U|$; that is, $O(|Q \times U|)$.

Pruning-based Linear Algorithm. While it is costly to enumerate all pairs of instances in $Q \times U$, intuitively most pairs in $Q \times U$ are possible to be removed without enumerating them. This may be done by using the local aR-trees of Q and U, respectively. Our algorithm is based on level-by-level synchronous traversal on the local aR-trres of Q and U. The example below gives the basic idea of the algorithm.

Basic Idea. Suppose that the root R_Q of local aR-tree of Q has 2 entries (E_1, E_2), and the root R_U of loca aR-tree of U has 2 entries (E'_1, E'_2). Totally, the 4 pairs of entries are depicted in Figure 6.5 where the two ends of each interval, corresponding to (E_i, E'_j), are $d^{lo}(E_i, E'_j)$ and $d^{up}(E_i, E'_j)$, respectively, and $w(E_i, E'_j)$ is also shown

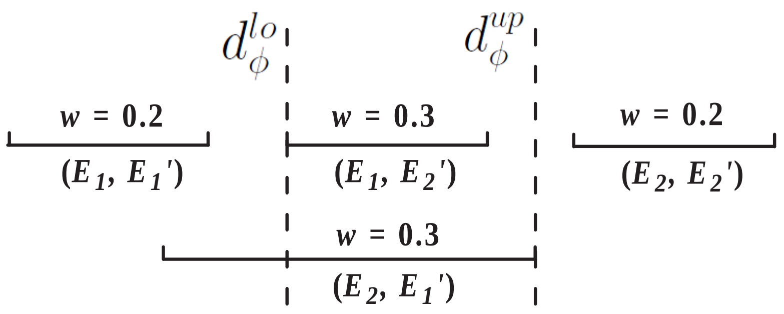


Figure 6.5: Prune Entries at the Current Level

as w. Assume that a *lower-bound* d_ϕ^{lo} and an *upper-bound* d_ϕ^{up} of $d_\phi(Q, U)$ are as what are depicted in Figure 6.5, respectively.

Since $d^{up}(E_1, E_1')$ is smaller than d_ϕ^{lo}, (E_1, E_1') can be removed. The pair of (E_2, E_2') can also be removed because $d^{lo}(E_2, E_2')$ is larger than d_ϕ^{up}. Consequently, we only focus on 2 pairs of entries, (E_1, E_2') and $(E2, E_1')$, in the next level iteration. As the distance of any pair of instances in (E_1, E_1') is guaranteed to be smaller than $d_\phi(Q, U)$ and the total weight of (E_1, E_1') is 0.2, from the next level we only need to find the $(\phi - 0.2)$-quantile distances in the remaining pairs of instances. This is the basic idea of our algorithm.

Algorithm Description. We outline our algorithm in Algorithm 6.3 below in a recursive fashion. Note that the input of the algorithm is a collection of pairs of entries - initially, $R_Q \times R_U$ where R_Q (R_U) is the set of the entries in the root of the local aR-tree of Q (U).

In Algorithm 6.3, lines 7 and 8 remove the pairs, with their maximum distances smaller than d_ϕ^{lo} or minimum distances larger than d_ϕ^{up}, from T_1 (i.e. no further exploring). Lines 9 and 10 cumulatively record the total weights θ of removed pairs of entries with the maximum distance smaller than d_ϕ^{lo}. Lines 11 and 12 enumerate all the remaining pairs of entries in the next level for the next iteration; this will be shown in the example below. To ensure the correctness, in the next iteration,

Algorithm 6.3 QUANTILE-DISTANCE ($R_Q \times R_U$, ϕ)

```
Input: $R_Q \times R_U$; $\phi$: $0 < \phi \leq 1$
Output: $d_\phi(Q, U)$
  $\theta := 0$; $T_1 := \emptyset$; $T_2 := \emptyset$;
  if $R_Q \times R_U$ only contains leaf entries then
      call Algorithm 6.1 on $R_Q \times R_U$ and $\phi$;
  else
      Calculating $d^{lo}_\phi$ and $d^{up}_\phi$ regarding $R_Q \times R_U$;
      for each $(E, E') \in R_Q \times R_U$ do
          if $d^{up}(E, E') \geq d^{lo}_\phi$ and $d^{lo}(E, E') \leq d^{up}_\phi$ then
              $T_1 := \{(E, E')\} \cup T_1$;
          else if $d^{up}(E, E') < d^{lo}_\phi$ then
              $\theta := \theta + w(E) \times w(E')$;
      for each $(E, E') \in T_1$ do
          $T_2 := T_2 \cup$ ENUMERATING$(E, E')$;
      QUANTILE-DISTANCE ($T_2$, $\phi - \theta$);
```

we compute the $(\phi - \theta)$-quantile distance from remaining pairs of instances.

Example 6.3. *Continue the example (Figure 6.5) in the part of Basic Idea. Using Algorithm 6.3, $T_1 = \{(E_1, E'_2), (E_2, E'_1)\}$ in the 1st iteration and $\theta = 0.2$.*

Assume that the child node, NODE(E_1), of R_Q corresponding to the entry E_1 has two entries $\{E_{1,1}, E_{1,2}\}$, NODE(E_2) contains $\{E_{2,1}, E_{2,2}\}$, NODE(E'_1) contains $\{E'_{1,1}, E'_{1,2}\}$, and NODE(E'_2) contains $\{E'_{2,1}, E'_{2,2}\}$. In Algorithm 6.3, ENUMERATING(E_1, E'_2) generates the 4 pairs of entries, $\{(E_{1,1}, E'_{2,1})$, $(E_{1,1}, E'_{2,2})$, $(E_{1,2}, E'_{2,1})$, $(E_{1,2}, E'_{2,2})\}$. Similarly, ENUMERATING(E_2, E'_1) generates the 4 pairs of entries, $\{(E_{2,1}, E'_{1,1})$, $(E_{2,1}, E'_{1,2})$, $(E_{2,2}, E'_{1,1})$, $(E_{2,2}, E'_{1,2})\}$. These 8 pairs (in T_2) together with $(\phi - 0.2)$ are sent to the next iteration - QUANTILE-DISTANCE (T_2,

$\phi - 0.2$). $\square$

Remark 5. *If $\{E_1, E_2\}$ are the leafs (i.e. points), then they have no corresponding children nodes. In this case, ENUMERATING(E_1, E'_2) and ENUMERATING(E_2, E'_1) generate 4 pairs of entries in total: $\{(E_1, E'_{2,1})$, $(E_1, E'_{2,2})$, $(E_2, E'_{1,1})$, $(E_2, E'_{1,2})\}$. Similarly, if $\{E'_1, E'_2\}$ are the leafs, then the following 4 pairs of entires are generated for the next iteration: $\{(E_{1,1}, E'_2)$, $(E_{1,2}, E'_2)$, $(E_{2,1}, E'_1)$, $(E_{2,2}, E'_1)\}$.* $\square$

Calculating d^{lo}_{ϕ} and d^{up}_{ϕ}. In Algorithm 6.3, at each iteration we need to calculate d^{lo}_{ϕ} and d^{up}_{ϕ} (line 5) except that all entries are at the leaf level. Assume that at the jth iteration, there are l pairs of entries left; that is, $T_2 = \{t_i \mid 1 \leq i \leq l\}$ - each t_i with the form (E, E') where E is an entry from the local aR-tree of Q and E' is an entry from the local ar-tree of U. Recall that the maximum distance $d^{up}(t_i)$ and the minimum distance $d^{lo}(t_i)$ are defined on a pair t_i of entries in Section 6.2. Suppose that in the current iteration, we want to compute the ϕ'-quantile distance $d_{\phi'}(\mathcal{T})$ in $\mathcal{T}$ where $\mathcal{T}$ denotes the collection of pairs of instances each of which has an element in T_2 as the ancestor; that is, for each pair of instances $(q, u) \in \mathcal{T}$, $\exists (E, E') \in T_2$ such that E contains q and E' contains u.

Let the ϕ'-quantile of T_2, regarding the search key $d^{lo}(t_i)$, be (EL, EL'), and the ϕ'-quantile of T_2, regarding the search key $d^{up}(t_i)$, be denoted by (EU, EU').

Theorem 6.1. *$d^{lo}(EL, EL') \leq d_{\phi'}(\mathcal{T}) \leq d^{up}(EU, EU')$.*

Proof. According to the definition of the ϕ'-quantile distance, it is immediate that $\sum_{d^{lo}(t) \leq d_{\phi'}(\mathcal{T}), t \in T_2} w(t) \geq \phi'$; consequently, $d^{lo}(EL, EL') \leq d_{\phi'}(\mathcal{T})$.

Similarly, according to the definition of ϕ'-quantile distance, $\sum_{d^{up}(t) < d_{\phi'}(\mathcal{T}), t \in T_2} w(t) < \phi'$. Therefore, $d_{\phi'}(\mathcal{T}) \leq d^{up}(EU, EU')$. $\square$

Theorem 6.1 implies that $d^{lo}(EL, EL')$ and $d^{up}(EU, EU')$ are a lower-bound and an upper-bound, respectively, of $d_{\phi'}(\mathcal{T})$. Thus, in line 5 of Algorithm 6.3, we calculate $d^{lo}(EL, EL')$ and $d^{up}(EU, EU')$. Clearly, this can be done by Algorithm 6.1 in linear time.

Time Complexity. In each iteration, our algorithm is linear regarding $|T_2|$; that is, $O(|T_2|)$. Since the total entries in the local aR-trees of Q and U are $(|Q|)$ and $O(|U|)$, respectively, Algorithm 6.3 runes in linear time regarding $|Q \times U|$; that is, $O(|Q \times U|)$.

Correctness. The following theorem can be immediately verified based on the definition of ϕ-quantile distance.

Theorem 6.2. *Let θ denote the total weights of the pairs of entries so far pruned by d^{lo}_{ϕ} at each iteration. Then, $d_{\phi-\theta}(\mathcal{T}) = d_{\phi}(Q, U)$ where $\mathcal{T}$ consists of all remaining pairs of instances after the current iteration.*

Theorem 6.1 and Theorem 6.2 imply that Algorithm 6.3 is correct; that is, it can produce $d_{\phi}(Q, U)$.

Filtering while Enumerating. Algorithm 6.3 can be improved when enumerating children pairs in line 12 - ENUMERATING(E, E'). For every enumerated pair t of children entries, before adding to T_2 we check if it can be pruned by the current distance lower and upper bounds. Then T_2 keeps only the remaining pairs of children entries for the next iteration. Note that in θ, we also include the total weights of children pairs pruned by the current lower bound d^{lo}_{ϕ}.

Example 6.4. *Continue Example 6.3. The enumerated* 8 *pairs are depicted in Figure 6.6. The pair $(E_{2,1}, E'_{1,1})$ with the weight 0.1 is pruned by the current d^{lo}_{ϕ}. Consequently, the remaining* 7 *pairs (put in T_2) and $(\phi - 0.2 - 0.1)$ are used to call QUANTILE-DISTANCE () for the next iteration.*

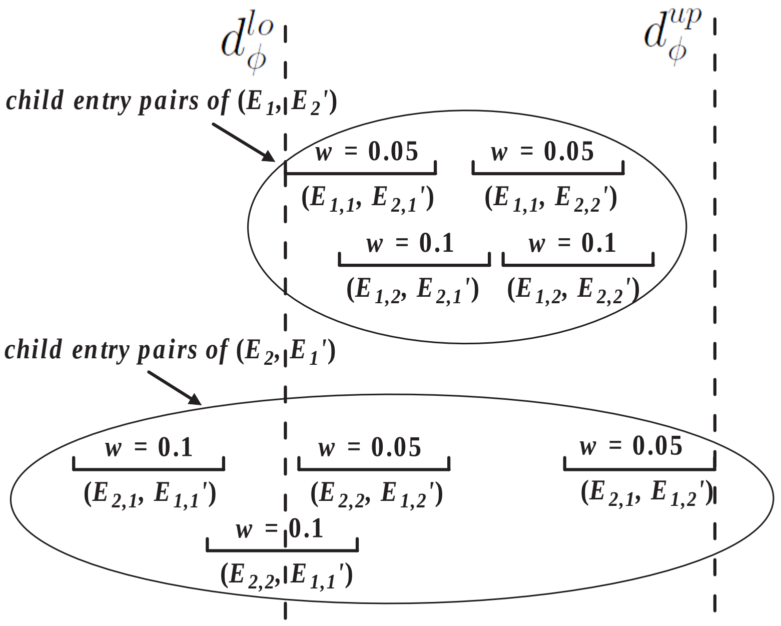


Figure 6.6: Filtering while Enumerating

Note that the time complexity of Algorithm 6.3, by adding the technique "Filtering while Enumerating", remains the same $O(|Q| \times |U|)$.

6.3.2 Refinement Algorithm

In the seeding phase, after $d_\phi(Q, U)$ is calculated for each U of chosen K objects. A max_heap maintains the K objects based on their ϕ-quantile distances; γ_K is the maximum of these K ϕ-quantile distances and sits on the top. Our refinement algorithm for generating the final result for ϕ-quantile KNN is outlined below in Algorithm 6.4. In the algorithm, we effectively use γ_K and ϕ to prune as many entries, in the global R-tree on MBBs of objects and local aR-trees, as possible. Since "closer" objects have a better chance to be in the final result of KNN (thus a better chance to reduce γ_K), we traverse the global R-tree based on the priority that an entry E with the smallest minimum distance to the MBB of Q will be visited first. This can be done by maintaining a heap H on the currently extended entries.

Algorithm 6.4 Refinement

```
1: while $H \neq \emptyset$ do
2:     $E$ := deheap($H$);
3:     if not PRUNED1($Q, E$) then
4:         if $E$ is an intermediate entry then
5:             add MBBs of the child entries to $H$;
6:         else
7:             if not PRUNED2($Q, E$) then
8:                 call Algorithm 6.3 to compute $d_\phi(E, Q)$;
9:                 if $d_\phi(Q, E) < \gamma_K$ then
10:                    update current KNN;
```

In Algorithm 6.4, we initially load to H the entries MBBs in the root node of the global R-tree. Then we iteratively apply PRUNED1(E, Q) to the heap top E by using the pruning rules below in Section 6.3.3 for the global R-tree. If E cannot be pruned (i.e. PRUNED1(Q, E) returns FALSE), then we add to H the entries of the child node with E as the MBB when E is an intermediate entry. When E cannot be pruned and E is the MBB of an object U, we apply the pruning rules below in Section 6.3.3 on an individual object, PRUNED2(Q, E), to prune U (PRUNED2(Q, E) returns TRUE if pruned) or "trim" the entries in the local aR-tree of U.

6.3.3 Pruning Rules

Pruning Rules 1 and 2 attempt to prune an entry in the global R-tree, while Pruning Rule 3 is to further examine the "details" of a remaining object using its local aR-tree.

Pruning an Entry E of Global R-tree. Pruning Rules 1 and 2 below use Q to prune an entry E of the global R-tree. The correctness of Pruning Rule 1 is immediate.

Pruning Rule 1. (*Distance based:*) If $d^{lo}(E_Q, E) \geq \gamma_K$, then E can be pruned where E_Q is the MBB of Q and E is an entry of the global R-tree. (E is pruned means that all objects indexed by E can be pruned).

Note that the minimum distance between two MBBs is defined in Section 6.2 and can be calculated in constant time. Next, we present Pruning Rule 2.

Regarding the local aR-tree aR_Q of Q, a set Γ of entries in aR_Q is a γ_K*-cover* of aR_Q if 1) there are no 2 entries in Γ with the descendent relationship, 2) for each $E_i \in \Gamma$, $d^{lo}(E_i, E) \leq \gamma_K$, and 3) for each entry E' which is not an ancestor nor a descendent of any entry in Γ, $d^{lo}(E', E) > \gamma_K$. The following theorem is immediate from the definition of ϕ-quantile distance.

Theorem 6.3. *Let Γ be a γ_K-cover of R_Q. If $\sum_{E' \in \Gamma} w(E') \leq \phi$, then for each $U \in E$, $d_\phi(Q, U) \geq \gamma_K$.*

Clearly, if we can find a γ_K-cover satisfying the condition in Theorem 6.3, then E can be pruned.

Pruning Rule 2. (*Weights based:*) If there is a γ_K-cover Γ with $\sum_{E' \in \Gamma} w(E') \leq \phi$, then E can be pruned.

Note that there could be many γ_K-covers as shown in Example 6.5.

Example 6.5. *As depicted in Figure 6.7, the γ_K-covers can be $\{E_1, E_2\}$, $\{E_1, E_{2,3}\}$, $\{E_{1,3}, E_2\}$, $\{E_{1,3}, E_{2,3}\}$. If $E_{1,3}$ and $E_{2,3}$ have child entries, more alternatives could be enumerated. They possibly have different total weights.*

A γ_K-cover Γ is *minimum* if $\sum_{E' \in \Gamma} w(E')$ is minimized. In example 6.5, $\{E_{1,3}, E_{2,3}\}$ has the smallest weight among those 4 covers. Clearly, the minimum

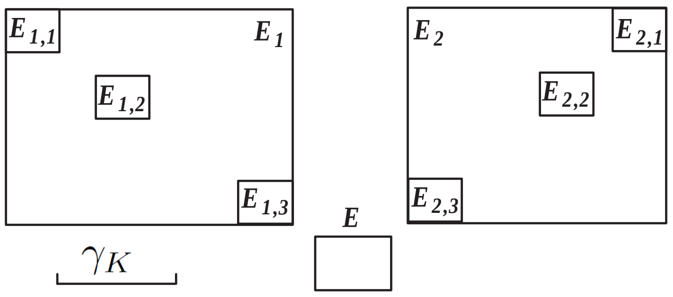


Figure 6.7: γ_K-Cover

γ_K has the maximal pruning power since $\sum_{E' \in \Gamma} w(E')$ is minimized.

Executing Pruning Rule 2. Although a minimum γ_K-cover can be computed by traversing aR_Q level-by-level from the root, we will not always try to get a minimum γ_K-cover if E can be pruned earlier. We visit aR_Q level-by-level from the root. At each level i, we generate a todo list TD_i (initially $\emptyset$), and remove/trim the child entries E' of the entries in TD_{i-1}, if $d^{lo}(E', E) > \gamma_K$. For each remaining child entry E' (not trimmed), E' with $d^{up}(E', E) \leq \gamma_K$ will not be extended at the next level since all its decedent entries always have their minimum (and maximum) distances not greater that γ_K - we cumulate $w(E')$ in Δ; E' with $d^{up}(E', E) > \gamma_K$ will be extended in the next level for further trimming (thus, it is put into TD_i). E is pruned and we terminate the execution of Pruning Rule 2 if the value of Δ plus the total weights of the entries in TD_i is not greater than ϕ. Note that if E cannot be pruned, then the execution terminates if either the the current TD_i is empty or at the leaf level. Moreover, at the root level (i.e. $i = 1$), we assume that TD_0 consists of the MBB E_Q of Q.

Clearly, the execution of Pruning Rule 2 terminates if E is pruned or the minimum value of γ_K-cover is obtained (Δ + the total weights in current TD). If E is the MBB of an object U (i.e. corresponds to U) and U cannot be removed, then we record the obtained total weights of the minimum γ_K-cover in Δ_Q and record its trimmed aR-local tree by $aR_{Q,trim}$. Δ_Q will be used in the next pruning rule, and $aR_{Q,trim}$ will be used in line 8 to call Algorithm 6.3 if U cannot be pruned by the next pruning rule.

Example 6.6. *Continue Example 6.5 regarding Figure 6.7. Suppose that the root of aR_Q contains entires E_1 and E_2. $TD_0 = \{E_Q\}$. At the root level, we obtain $TD_1 = \{E_1, E_2\}$ regarding the depicted γ_k. At the next level, $E_{1,1}$, $E_{1,2}$, $E_{2,2}$, $E_{2,1}$ are trimmed; consequently, $TD_2 = \{E_{1,3}, E_{2,3}\}$ and $\Delta = 0$ if $d^{up}(E_{1,3}, E) > \gamma_K$ and $d^{up}(E_{2,3}, E) > \gamma_K$. If $w(TD_2) < \phi$, then E will be pruned; otherwise we go to the next level for further exploring.*

In case that $d^{up}(E_{1,3}, E) \leq \gamma_K$ and $d^{up}(E_{2,3}, E) \leq \gamma_K$, TD_2 remains $\emptyset$ and $\Delta = w(E_{1,3}) + w(E_{2,3})$. If $\Delta > \phi$ then E cannot be pruned. Since $TD_2 = \emptyset$, the execution of Pruning Rule 2 terminates. If E is an object, then we record Δ_Q and $aR_{Q,trim}$. Here, $\Delta_Q = w(E_{1,3}) + w(E_{2,3})$, and in $aR_{Q,trim}$, $E_{1,1}$, $E_{1,2}$, $E_{2,1}$, $E_{2,3}$ are pruned/trimmed. $\square$

Remark 6. *When we trim/remove entries of R-trees, we do a "logic" removal by commenting them out.*

PRUNED1(Q, E). For each entry, we first check Pruning Rule 1 - PRUNED1(Q, E) returns TRUE if E is pruned. If E cannot be pruned by Pruning Rule 1, then we invoke the above execution of Pruning Rule 2; PRUNED1(Q, E) returns TRUE if E is pruned.

Trimming the Local aR-Tree of U. Before conducting the computation of ϕ-quantile distance by Algorithm 6.3, we first trim the entries of the local aR-tree by γ_K. We conduct this in a level-by-level fashion from the root of the local aR-tree in the same way as the execution of Pruning Rule 2 except that we swap the role Q with U; that is, Q becomes E, and U becomes Q in the execution of Pruning

Rule 2. At each level i of aR-tree of U, we check the flowing pruning rule.

Pruning Rule 3. (*Using Local aR-tree:*) If $(\Delta + w(TD_i)) \times \Delta_Q \leq \phi$, then U can be pruned. [4]

Proof. From the definition of ϕ-quantile distance, it is immediate that if $(\Delta + w(TD_i)) \times \Delta_Q \leq \phi$, then $d_\phi(Q, U) \geq \gamma_K$. □

PRUNED2(Q, E). As described above, the execution of Pruning Rule 3 is the same as the execution of Pruning Rule 2 except that we swap the roles of Q and U and check Pruning Rule 3 instead of Pruning 2 at each level. PRUNED2(Q, E) terminates and returns TRUE if E is pruned; otherwise, PRUNED2(Q, E) terminates at the leaf-level (or $TD_i = \emptyset$) and returns FALSE. When PRUNED1(Q, E) returns FALSE, $R_{Q,trim} \times R_{U,trim}$ is used as the input of Algorithm 6.3 for computing $d_\phi(Q, U)$ instead of using $R_Q \times R_U$. Here, $R_{Q,trim}$ ($R_{U,trim}$) consists of the untrimmed entries at the root of $aR_{Q,trim}$ ($aR_{U,trim}$). Note that in Algorithm 6.3, level-by-level we use only untrimmed entries from both $aR_{Q,trim}$ and $aR_{U,trim}$. We can further speed-up the computation by visiting only the intermediate nodes, in $aR_{Q,trim}$ and $aR_{Q,trim}$, respectively, with more than one child.

The correctness of Algorithm 6.4 immediately follows from the theorems and pruning rules. Note that when Algorithm 6.3 is invoked, at each iteration we use the minimum value of γ_K and the obtained upper-bound d_ϕ^{up} as an upper-bound.

[4]Here, Δ_Q is obtained as the weight of the minimum γ_K-cover of the local aR-tree of Q. Δ and TD_i are recorded when execute the Pruning Rule 2 at level i and swap the roles of Q and U as described above.

6.4 ϕ-quantile Group-base KNN

Our algorithm for solving the ϕ-quantile group-base KNN ($\phi \in (0, 1]$) (defined in Section 6.1.1) also follows the seeding-refinement framework, Algorithm 6.2. For the seeding-phase, firstly we show that computing a ϕ-quantile group-base distance $gbd_\phi(Q, U)$ between Q and U is NP-hard, and then an existing algorithm is employed with the approximation factor 2 to approximately compute $gbd_\phi(Q, U)$. In the refinement phase, 2 novel, effective pruning techniques are developed.

6.4.1 Computing ϕ-Quantile Group-base Distances

We first show that the *Knapsack Problem* can be converted to a special case of our problem.

Knapsack Problem. It is NP-complete and can be formally described below [GJ90].

INSTANCE: Finite set S, for each element $s \in S$, an integer size $c(s)$, and an integer value $v(s)$, and positive integers X and Y.

QUESTION: Is there a subset S' of S such that $\sum_{s\in S'} c(s) \leq X$ and $\sum_{s\in S'} v(s) \geq Y$.

NP-hardness. As defined in Section 6.1.1, the problem of computing $gbd_\phi(Q, U)$ can be stated below. Find a subset S' from $Q \times U$ such that $\sum_{(q,u)\in S'} w(q, u) \geq \phi$ and $\sum_{(q,u)\in S'} w(q, u)d(q, u)$ is minimized.

A special case of the problem of computing $gbd_\phi(Q, U)$ is that Q is a point with weight 1. In this case, each $w(u)$ $(= w(Q, u))$ may be arbitrarily assigned with the constraint $\sum_{u\in U} w(u) = 1$, and the location of each u can be chosen so that $w(Q, u)d(Q, u)$ equals any integer.

If we normalize the above Knapsack Problem by normalizing each $v(s)$ by

$\frac{v(s)}{\sum_{s' \in S} v(s')}$. Then the normalized version of Knapsack is also NP-complete. The *decision* problem of the above spacial case of computing $gbd_\phi(Q, U)$ is the same as the normalized Knapsack Problem. Consequently, the problem of computing $gbd_\phi(Q, U)$ is NP-hard.

Theorem 6.4. *The problem of computing $gbd_\phi(Q, U)$ is NP-hard.*

Approximately Computing $gbd_\phi(Q, U)$. If we want to maximize $\sum_{s \in S'} v(s)$ with respect to a given X in the Knapsack Problem, then there is PTAS; that is, a polynomial-time approximation scheme giving an approximate factor arbitrarily closer to 1. Nevertheless, there is no PTAS to approximately minimize $\sum_{s \in S'} c(s)$ regarding a given Y.

We adopt the approximate algorithm in [GJ00] for Knapsack Problem. It runs in time $O(m \log m)$, where m is the number of elements in S, with the approximation factor 2 for minimizing $\sum_{s \in S'} c(s)$ for a given Y. The algorithm can be immediately used to approximately compute $gbd_\phi(Q, U)$ if we treat $Q \times U$ as S; and for each $(q, u) \in Q \times U$, treat $w(q, u)$ as a v value and treat $w(q, u)d(q, u)$ as a c value in the Knapsack Problem. Let $aproxgbd_\phi(Q, U)$ denote the group distance output by the approximation algorithm. The following theorem is shown [GJ00].

Theorem 6.5. $1 \leq \frac{aproxgbd_\phi(Q,U)}{gbd_\phi(Q,U)} \leq 2$.

We briefly present the basic idea of the algorithm in [GJ00] while applying it to computing gbd_ϕ. It iteratively conducts 2 phases: Completion and Growing Seed-Set ST - initially $\emptyset$ ($w(ST)$ is always smaller than ϕ). We firstly sort $Q \times U$ increasingly based on $d(q, u)$. In the Completion phase, for each element (q, u) in the remaining $Q \times U$ with $w(q, u) + w(ST) \geq \phi$, 1) replace the current feasible solution S' if the total weighted distance in $ST \cup \{(q, u)\}$ is smaller than that in S', and 2) remove (q, u) from $Q \times U$. In Growing Seed-Set ST, move the 1st element from the remaining $Q \times U$ to ST. In each iteration, we first conduct Completion and then Growing Seed-Set; the algorithm terminates and outputs the total weighted distance in S' if there is no element left in the remaining $Q \times U$.

Example 6.7. *Suppose that* $\phi = 0.5$ *and* $Q \times U$ *contains* 4 *elements. To simplify the presentation, we present these* 4 *elements only by its (distance, weight):* $\{(1, 0.28)$, $(2, 0.12)$, $(3, 0.48)$, $(4, 0.12)\}$. *In our algorithm, we first sort the list increasingly based on the value of* $\frac{weight \times distance}{weight} = distance$.

In the 1st iteration, nothing is chosen in the Completion phase since all elements with weight less than 0.5*;* ST *becomes* $\{(1, 0.28)\}$ *and* $(1, 0.28)$ *is removed from* $Q \times U$ *in the Growing Seed-Set phase. In the 2nd iteration,* $S' = \{(1, 0.28), (3, 0.48)\}$ *is chosen as a feasible solution and* $(3, 0.48)$ *is removed* $Q \times U$ *in the Completion phase;* ST *grows to* $\{(1, 0.28), (2, 0.12)\}$ *and* $(2, 0.12)$ *is removed from* $Q \times U$ *since* $(2, 0.12)$ *was the 1st element. In the 3rd iteration, regarding Completion phase,* $(4, 0.12)$ *is removed from* $Q \times U$ *as* $w(ST) + 0.13 = 0.52 > 0.5$ *and* $\{(1, 0.28)$, $(2, 0.12)$, $(4, 0.12)\}$ *becomes* S' *as its total weighted distance (*1*) smaller than that (*1.72*) in* $S' = \{(1, 0.28), (3, 0.48)\}$. *Consequently,* 1 *is output as* $approxgbd_{0.5}(Q, U)$*; in this example it happens* $approxgbd_{0.5}(Q, U) = gbd(Q, U)$. □

Note that this approximate algorithm does not accommodate a pruning-based level-by-level computation of $gbd_{\phi}(Q, U)$ because it requires to access all elements.

6.4.2 Refinement

In the seeding phase, we use the above approximate algorithm to approximately compute $gbd_{\phi}(Q, U)$ between Q and each of the chosen K objects. The largest obtained $aproxgbd_{\phi}$ value is denoted as λ_K. The refinement algorithm follows the similar framework outlined in Algorithm 6.4 in Section 6.3.2 except that:

- In PRUNDE1(Q, E) we will use the pruning rules below.
- remove line 7.
- call the above algorithm to (approximately) compute $gbd_\phi(Q, U)$ instead of Algorithm 6.3.
- use $aproxgbd_\phi$ generated by the above approximate algorithm and λ_K to replace d_ϕ and γ_K, respectively.

In the group with its total weighted distance $gbd_\phi(Q, U)$, instances may be from many different entries of the local aR-tree of U. Consequently, it is not always possible to trim many entries (subtrees) from the local aR-tree as what we do for computing ϕ-quantile KNN. Thus, in our refinement algorithm we only develop pruning rules to prune entries in the global R-tree.

Pruning Rule 4. Suppose that E_Q is the MBB of Q. If $\phi \times d^L(E_Q, E) \geq \lambda_K$, then E is pruned from the global R-tree.

The next pruning rule is used at each level. Suppose that $L_k = \{E_i \mid 1 \leq i \leq l\}$ consists of all the entries at the level k of the local aR-tree of Q. Without loss of generality, we assume that L_k is sorted in the increasing order based on $d^L(E_i, E)$; that is, $d^L(E_{i1}, E) \leq d^L(E_{i2}, E)$ if $i1 < i2$. Let E_j denote the ϕ-quantile of L_k according to the search key $d^L(E_i, E)$ and the weight $w(E_i)$ of each element $E_i \in L_k$.

Pruning Rule 5. E is pruned if:

$$(\phi - \sum_{i=1}^{j-1} w(E_i))d^L(E_j, E) + \sum_{i=1}^{j-1}(w(E_i) \times d^L(E_i, E)) \geq \lambda_K.$$

Executing PRUNDE1(Q, E). For an E in the global R-tree, we first check Pruning Rule 4; this is done by constant time. If E cannot be pruned, then we traverse the local aR-tree of Q level-by-level from the root to test Pruning Rule

5. To test Pruning Rule 5 at each level k, we first need to sort L_k. The total time complexity for traversing the local aR-tree of Q to test Pruning Rule 5 is thus $O(|Q|)$.

Accuracy Guarantee. Our algorithm for solving ϕ-quantile group-base KNN has the following accuracy guarantee.

Theorem 6.6. *Suppose that for* $1 \leq i \leq k$, U_i *is ranked the top-ith in the exact* ϕ*-quantile group-base KNN, and* U'_i *is ranked the top-ith by our algorithms. Then for* $1 \leq i \leq k$, $gbd_\phi(Q, U_i) \leq aproxgbd_\phi(Q, U'_i) \leq 2gbd_\phi(Q, U_i)$.

Proof. First, it can be immediately verified that the object U pruned (i.e., the entry E containing U is pruned) by Pruning Rule 4 or Pruning 5 has the property that $gbd_\phi(Q, U) \geq \lambda_K$. From Theorem 6.6, it follows that for $1 \leq i \leq k$, $gbd_\phi(Q, U_i) \leq aproxgbd_\phi(Q, U'_i) \leq 2gbd_\phi(Q, U_i)$. □

Theorem 6.6 states that every ith group-base distance ($i \in [1, K]$) output by our algorithm is between $gbd_\phi(Q, U_i)$ and $2gbd_\phi(Q, U_i)$. Our experiment, nevertheless, indicates the error could be much smaller in practice.

6.5 Experimental Study

We report a thorough performance evaluation on the efficiency and effectiveness of our algorithms. In particular, we implement and evaluate the following techniques.

Q-KNN: Techniques presented in Section 6.3 to compute KNN based on a ϕ-quantile distance ($\phi \in (0, 1]$).

Naive Q-KNN: Remove the pruning rules from Q-KNN.

G-KNN: Techniques in Section 6.4 to compute KNN based on ϕ-quantile group-base distances.

Naive G-KNN: Remove the pruning rules from G-KNN.

All algorithms are implemented in C++ and compiled by GNU GCC. Experiments are conducted on PCs with Intel Xeon 2.4GHz dual CPU and 4G memory under Debian Linux. Our experiments are conducted on both real and synthetic datasets.

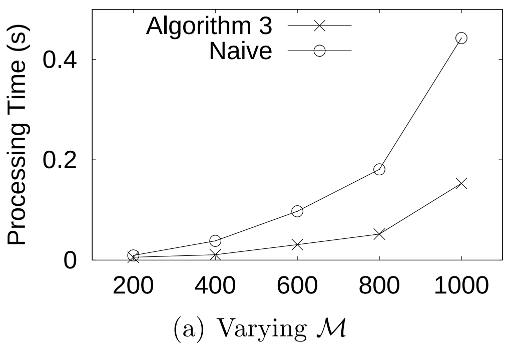
Real dataset is extracted from NBA players' game-by-game statistics (http://www.nba.com), containing 339,721 records of 1,313 players. Each player is treated as a multi-valued object where the statistics (score, assistance, rebound) of a player per game is treated as an instance with the equal weight (normalized).

Synthetic datasets are generated using the methodologies in [BKS01] regarding the following parameters. Dimensionality d varies from 2 to 5 with default value 3. Data domain in each dimension is $[0, 1]$. Number n of objects varies from $10,000$ to $50,000$ with default value $10,000$. Number m of instances per object follows a uniform distribution in [1, $\mathcal{M}$] where $\mathcal{M}$ varies from 400 to $2,000$ with the default value 400. The value K varies among 5, 10, 20, 30 and 40 with default value 10. The average length of object MBBs follows a *uniform* or *normal* distribution. In normal distribution, the length of MBB lies in the range $[0, h]$ with the expectation value $h/2$ and standard deviation 0.025; in uniform distribution, the length of MBBs uniformly spreads over [0, h] where h varies from 0.05 to 0.25 with default value 0.05 (i.e., 5% of the edge length of the whole data space). With the default setting, the total number of instances is about 2 millions.

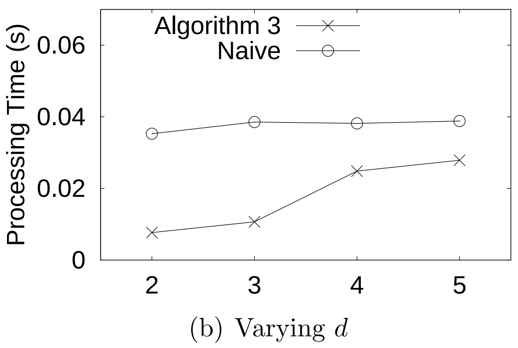
Centers of objects (objects' MBBs) follow either *uniform*, *normal* or *anti-correlated* distribution. Locations of instances in an object follow *uniform* or *normal* distribution. Weights assigned to each instance follow *uniform* or *normal* distribution. Table 6.2 summarizes the parameters used in our experiment where the default values are in **bold** font. *For each experiment, we randomly choose* 100 *objects from datasets as query objects and record the average performance. Note that default values will be used in our experiment unless otherwise specified.*

dimensionality d	2, **3**, 4, 5
number of objects $\mathcal{N}$	**10k**, 20k, 30k, 40k, 50k
edge length h	**0.05**, 0.1, 0.15, 0.2, 0.25
number of instances m	**400**, 600, 800, 1k, 2k
K	5, **10**, 15, 20, 30
ϕ	0.1, 0.3, **0.5**, 0.7, 0.9
object location	uniform, normal, **anti-correlated**
instance location	**uniform**, normal
weight distribution	uniform, **normal**
h distribution	**uniform**, normal

Table 6.2: Parameter Values.



(a) Varying $\mathcal{M}$



(b) Varying d

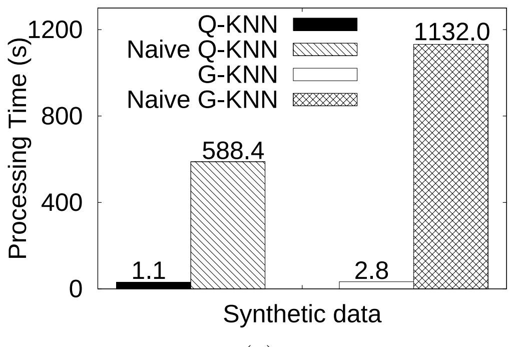
Figure 6.8: Time for Computing d_ϕ

6.5.1 Computing ϕ-Quantile Distance

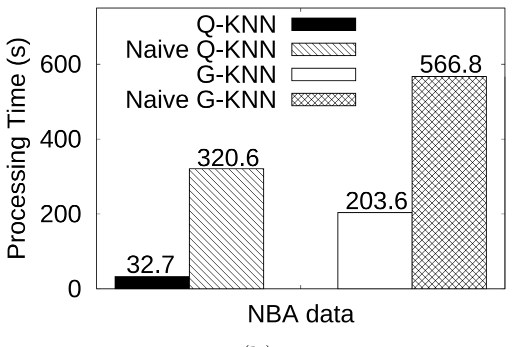
Figure 6.8 evaluates the efficiency of our technique, Algorithm 6.3, for computing a ϕ-quantile distance, against the naive algorithm described in Section 6.3.1. In our experiment, we randomly select 1000 pairs of objects from the datasets to test these 2 algorithms and report the average time by seconds. Figure 6.8(a) shows that our technique has more advantages when the number of instances increases. Figure 6.8(b) shows that the advantage of using Algorithm 6.3 gets lower when dimensionality increases. This is because that the pruning costs in Algorithm 6.3 are proportional to the dimensionality. When dimensionality increases, more pruning overheads are involved. Nevertheless, Figure 6.8 indicates Algorithm 6.3 significantly outperforms the naive algorithm. Therefore, we always use Algorithm 6.3 in the remaining experiments. Note that we did not evaluate the techniques in [YMT06] since they are not generally applicable to our problem.

6.5.2 Overall Performance

Figure 6.9 reports the results of the evaluation on processing time of Q-KNN, Naive Q-KNN, G-KNN, Naive G-KNN over real and synthetic datasets. As shown, Q-KNN and G-KNN are much more efficient than their naive versions (i.e. without using pruning techniques in the refinement phase) - upto 2 orders of magnitude. The improvement is less significant over NBA data. This is because in NBA dataset, objects' MBB sizes are very large relative to the whole data space; this gives very high overlapping degree among objects' MBBs. Thus less objects can be pruned during query processing.



(a)



(b)

Figure 6.9: Overall Performance

We further evaluate the pruning powers in the refinement phase by conducting the following experiment. Regarding the ϕ-quantile KNN, we examine the running time of Naive Q-KNN, Naive Q-KNN with the Pruning Rule 1 (P1), Naive Q-KNN with the Pruning Rules 1 and 2 (P1-2), and the Naive Q-KNN with the Pruning Rules 1, 2, and 3 (P1-3, that is, Q-KNN). Similarly, for ϕ-quantile group-base KNN, Naive G-KNN, Naive G-KNN with the Pruning Rule 4 (P4), and Naive G-KNN with the Pruning Rules 4 and 5 (P4-5, that is, G-KNN) are examined. The evaluation results are depicted in Figure 6.10. It shows that all these pruning rules are very effective and efficient. These 2 experiments indicate that Q-KNN and G-KNN are much more efficient than Naive Q-KNN and Naive G-KNN, respectively. Thus, in the rest of experiments we will no longer evaluate Naive Q-KNN and Naive G-KNN.

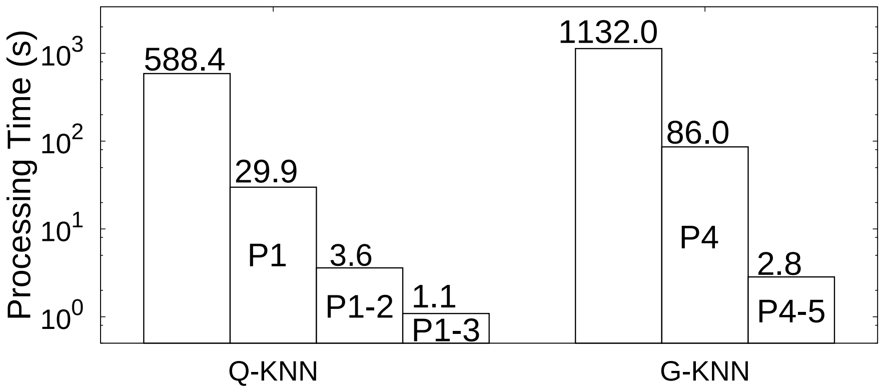


Figure 6.10: Pruning Powers

6.5.3 Accuracy

To evaluate the accuracy of G-KNN, we use two error measures. The first is the average distance error ratio. For $1 \leq i \leq K$, $approx(i)$ denotes the group-based distance of the top-ith object output by G-KNN, and $exact(i)$ denotes the group-based distance of the top-ith object in the exact solution.

$$err\_ratio = \frac{\sum_{i=1}^{K} \frac{|approx(i) - exact(i)|}{exact(i)}}{K}$$

The second measure records the "misplaced" ratio. For $1 \leq i \leq K$, if the ith object in the exact solution is not the same as the ith object in the solution output by G-KNN, then $mp(i) = 1$.

$$mp\_ratio = \frac{\sum_{i=1}^{K} mp(i)}{K}$$

As the ϕ-quantile group-base KNN is NP-hard and no efficient algorithm exists, we generate the exact solutions by a trivial exhaustive search - it is exponential and very slow. We conduct a very small scale experiment as follows. Each object, including query object, has 4 instances; there are total 100 objects. Others all use the default settings in Table 6.2. Table 6.3 reports the evaluation results when object distribution varies, while Table 6.4 reports the results when the distribution of weights varies. Both demonstrate G-KNN is highly accurate and more accurate than the theoretical guarantee in Theorem 6.6; that is, err_ratio is much smaller than 2.

	err_ratio	mp_ratio
anti	0.015	0.02
unif	0.013	0.02
norm	0.015	0.04

Table 6.3: Vary Objects Distribution

	err_ratio	mp_ratio
unif	0	0
norm	0.015	0.02

Table 6.4: Vary Weight Distribution

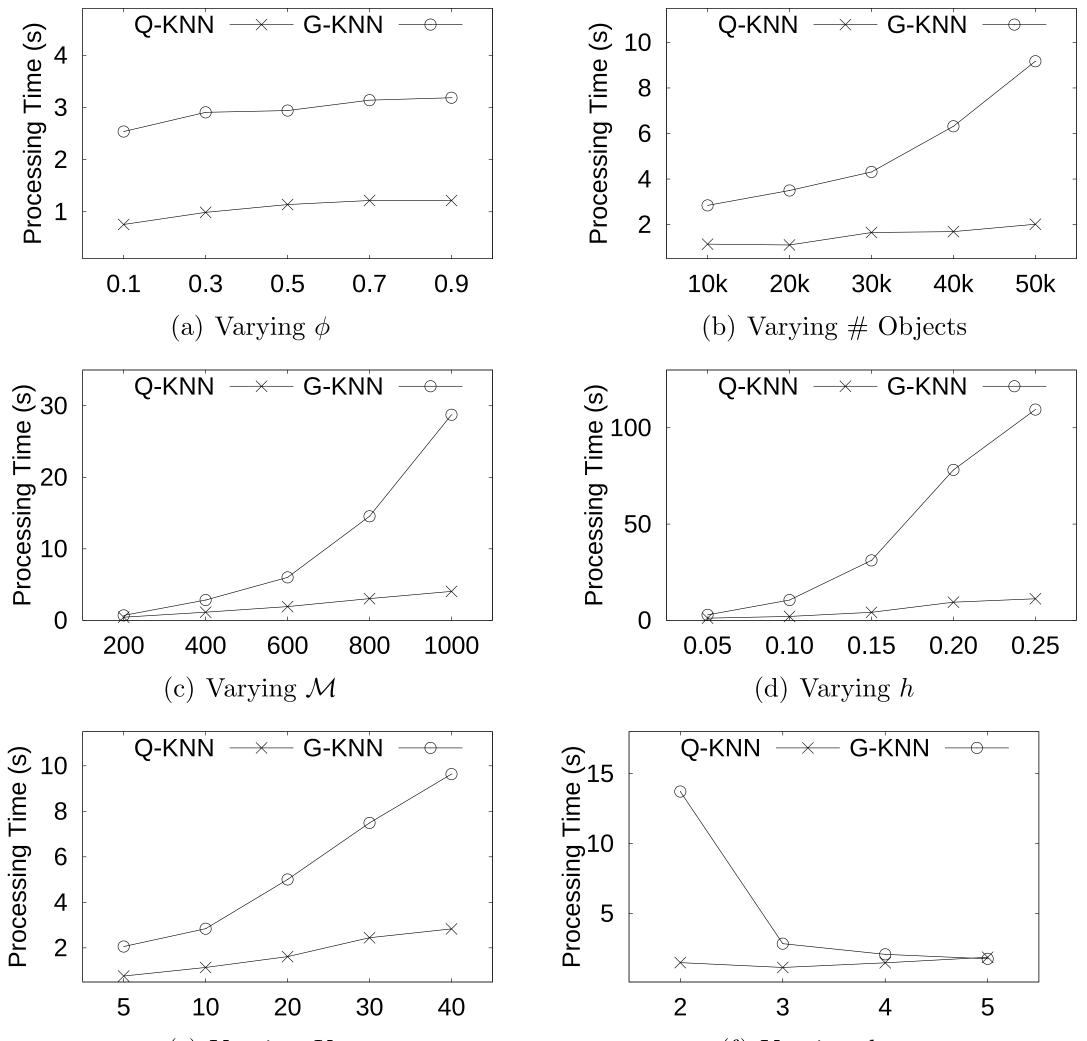
6.5.4 Evaluating Impacts by Different Settings

Distributions. We evaluate possible impacts on algorithm efficiency by distributions of centers of objects, locations of instance, edge lengths of object MBBs, and weights. The results (time in seconds) for Q-KNN and G-KNN are reported in Table 6.5, respectively. They demonstrate that Q-KNN is not quite sensitive to various distributions but G-KNN is quite sensitive towards different distributions. This is because of the nature of ϕ-quantile group-base distance - group-base. Note that it is only meaningful for object locations to have anti-distributions; consequently, we do not evaluate other distributions using anti. Moreover, the experiment shows anti always leads to more computation time; this is the reason why we use anti as a default setting for locations.

	Q-KNN			G-KNN		
	unif	norm	anti	unif	norm	anti
object_loc	0.9(s)	0.8(s)	1.1(s)	2.3(s)	2.0(s)	2.8(s)
MBB_length	1.1(s)	1.2(s)	*	2.8(s)	2.9(s)	*
instance_loc	1.1(s)	1.0(s)	*	2.8(s)	2.3(s)	*
weights	1.1(s)	1.1(s)	*	2.0(s)	2.8(s)	*

Table 6.5: Various Distributions

Impacts by Other Settings. In the next set of experiments, we study the scalability of our algorithms regarding different ϕ-values, number of objects, number of instances ($\mathcal{M}$), lengths of MBB edges (h), K, and the dimensionality d. In our experiments, we record the average running time per query for each algorithm. While Q-KNN and G-KNN are not quite sensitive to different ϕ-values due to the nature of the techniques developed, they are quite sensitive to the other settings especially G-KNN. The techniques in G-KNN do not have pruning rules for trimming object entries and the distance computation techniques of G-KNN do not have any pruning rules either. Thus, G-KNN is very sensitive to the increment of number of objects, number of instances, MBB lengths, and K. It is interesting to note that G-KNN runs faster when the dimensionality d increases. This suggests that G-KNN prunes more objects in the refinement phase when d increases. A possible reason is that when we fix the MBB edge length, the average area of MBBs gets smaller related to the whole data space; consequently, Pruning Rules 4 and 5 are more effective as they are group-based (thus, area based).



(a) Varying ϕ (b) Varying # Objects

(c) Varying $\mathcal{M}$ (d) Varying h

(e) Varying K (f) Varying d

Figure 6.11: Other Settings

6.5.5 Summary

Our performance evaluation indicates that Q-KNN is very efficient and scalable. Although ϕ-quantile group-base KNN is inherently more complex, our G-KNN techniques still perform quite efficiently. Furthermore, G-KNN is highly accurate and performs much more accurate than the theoretical bound.

6.6 Conclusion

In this chapter, we investigate the problem of KNN search over multi-valued objects. In particular, we use the quantile paradigm to retrieve KNN sensitive to the relative distribution among multi-valued objects. Two quantile KNN models have been proposed. One is based on a ϕ-quantile ranking score (e.g. median score) and another is based on the overall ranking score of the ϕ-quantile best population. We show that the second KNN problem is NP-hard. A set of efficient, novel techniques have been developed to process the first quantile KNN problem. Due to the NP-hardness of the second KNN problem, efficient approximate techniques with approximate factor 2 are presented. We conduct extensive experiments to illustrate the efficiency and effectiveness of our proposed techniques.

The current algorithms developed are based on main-memory computation. Although they can be immediately extended to support I/O involved computation, a possible future work may investigate I/O efficient techniques for these 2 KNN problems.

Chapter 7

Effectively Indexing the Uncertain Space [1]

Range search over uncertain data is important in query processing and data mining which have many applications. As an example, a server monitors a set of taxis equipped with GPS and location information of each taxi is sent back to the server every 5 minutes. Based on this periodically updated location information and other factors like velocity constraint, at each time stamp the location of a taxi is within a circle until next update arrives. The server may issue queries like "find taxis which are currently within 10 kilometers from the city tower". Since the location is not exact, a taxi may satisfy this query partially, as shown in Figure 7.1. The grey circles represent uncertain region of taxis while the transparent circle is the query region. While A definitely satisfies the query, B and C *probably* satisfy it, which means they are within the query region with a *probability*. This probability can be intuitively computed based on the intersection between one uncertain region and

[1]The techniques presented in this chapter originally appear in the paper "Effectively Indexing the Uncertain Space", Ying Zhang, Xuemin Lin, Wenjie Zhang, Jianmin Wang and Qianlu Lin, *accepted by IEEE Transactions on Data Engineering (TKDE), 2009*

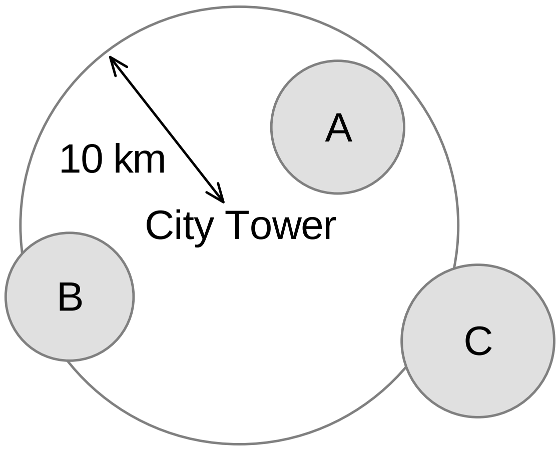


Figure 7.1: Taxis within 10 km from the City Tower

the query region, also the specific probability density function (PDF) information inside each taxi's uncertain region. Results with low probability values are often of no interest to users and a probability threshold is sometimes given beforehand to return results with probability no less than this threshold only.

Continuing with the example of monitoring taxis in Figure 7.1. In some cases specific identification of taxis are not necessary and only aggregate information is required, such as "how many taxis are currently inside city". Taxis with uncertain region partially inside city will also be considered probabilistically; Ranking based on probability is another way to handle possible results besides the threshold based fashion. To schedule the taxis, the server may retrieve 10 available taxis satisfying "within distance at most 5km from Four Seasons Hotel" with highest probability. Such a query is called a probabilistic top-k range query.

Range search is also a key component in the *filtering* phase of many queries in mining uncertain data such as spatial similarity join and k nearest neighbour query. Particularly, the spatial similarity join is essential to identify pairwise similar objects represented by uncertain multidimensional data. And k nearest neighbour query plays an important role in the study of spatial clustering and machine learning.

There are two types of techniques for indexing uncertain data with arbitrary PDF. The first type is R-Tree based index [CKP04, KHKH07, KKPR06, LBR$^+$07, SMP$^+$07]. More specifically, the uncertain region of multidimensional uncertain objects are grouped by R-Tree where each data unit is the minimum bounding rectangle(MBR) of a PDF. The drawback of this approach is that the uncertain region of an object is considered as an atomic unit, which leads to a poor performance for the probabilistic threshold based queries when individual MBRs are large. The second type of index is based on *probabilistically constrained regions* (PCRs) [CC07, TCX$^+$05]. The uncertain region of an object is partitioned with respect to a set of probability values. Partitioning results are then organized into a R-Tree style structure named U-Tree. U-Tree significantly outperforms *uncertain region* based index by utilizing probability thresholds in range query processing. While U-Tree supports the range queries with rectangular regions aligning the dimensional axes of data space well, it may not always provide a good support to range queries with non-rectangular regions or rectangular regions not aligning to axes. Details and analysis of existing indexes will be introduced in Section 7.1.

Motivated by these facts, in this chapter we study the problem of indexing uncertain data to support queries that require efficient range query processing. Contributions can be summarized as follows.

- A space-efficient index structure for organising multidimensional uncertain objects, UI-Tree, is proposed. UI-Tree can support arbitrary PDF of uncertain objects.

- We develop efficient solutions for various types of queries based on UI-Tree, including range query, size estimation of range query, probabilistic top-k range query and similarity join.

- We provide rigorous analysis to estimate the filtering capacity of UI-Tree.
- Extensive experiments over real and synthetic data sets are conducted to demonstrate the efficiency and scalability of UI-Tree compared with other state-of-the-art techniques.

The rest of the chapter is organized as follows. We formally define the problem and provide background information in Section 7.1. Section 7.2 presents the UI-Tree index structure. Section 7.3 applies UI-Tree to support different types of queries. Results of comprehensive performance studies are discussed in Section 7.4. In Section 7.5, we conclude the chapter.

7.1 Background

In Section 7.1.1, we first formally define the model of multidimensional uncertain objects and queries studied in the chapter. These are followed by the problem statement. Existing indexing approaches are reviewed in Section 7.1.2. Table 7.1 below summaries mathematical notations used throughout the chapter.

7.1.1 Problem Definition

Points referred in this chapter, by default, are in d-dimensional numerical space $D = \{D_1, \dots, D_d\}$ where D_i denotes the i-th dimension. A multidimensional uncertain object U can be regarded as a point whose location might appear at some locations with certain probabilities. Each possible appearance of the object is regarded as an instance of the uncertain object. Whenever there is no ambiguity, for instance u, we use u and $u.p$ to represent the location(point) of the instance and its appearance probability respectively. For presentation simplicity, we use “uncertain object” to represent “multi-dimensional uncertain object”.

Notation	Definition
U,V	uncertain objects
$\mathcal{U},\mathcal{V}$	set of uncertain objects
n	the number of uncertain objects in data set
Q (Q_r)	range query (query region)
θ	probabilistic threshold
$P_{app}(Q,U)$	the appearance probability of U w.r.t Q
l	number of partitions for each uncertain object
$w(w_p, w_{list})$	a *word* (probability value, posting list)
$A(w)(A(Q))$	the area of $w_{mbr}(Q_r)$
$WA(w)$	the weighted area of a word
$f_i(f_l)$	fan-out of non-leaf(leaf) node in UI-Tree
$t_{w,U}$	tuple from w_{list} with oid U
$P(t_{w,U})$	probability of tuple $t_{w,U}$
C	Candidate set
m	*merge* factor

Table 7.1: The Summary of Notations.

An uncertain object can be described either *continuously* or *discretely.* In the *continuous* case, an uncertain object U is described by its PDF $U.pdf$ and uncertain region U_r. The appearance probability of an instance $x \in U_r$ is $U.pdf(x)$ and $\int_{x\in U_r} U.pdf(x)dx = 1$. In the *discrete* case, an uncertain object U consists of a set of instances $u_1, \ldots, u_m$ where u_i appears with probability $u_i.p$ and $\sum_{u\in U} u.p = 1$. For the presentation simplicity, we only discuss the *continuous* cases in the following part of the chapter as discrete cases can be easily mapped to *continuous* cases.

Before defining range queries over uncertain data, we first define appearance probability of an uncertain object with respect to the query region. Because of the uncertainty of the location of an object, it may be no longer meaningful to simply declare that it appears or does not appear in the query region. For a given query Q with query region Q_r and uncertain object U, we use $P_{app}(U,Q)$ to represent the probability that U falls in Q_r. $P_{app}(U,Q)$ is defined as follows.

$$P_{app}(U,Q) = \int_{x\in U_r\cap Q_r} U.pdf(x)dx.$$

Usually, query results with low probabilities are of no interest to users. Many queries studied in the literature are accompanied with a user defined probabilistic threshold θ which reflects the requirements or confidence level of the user. Following is the definition of probabilistic threshold range query[CXP$^+$04, TCX$^+$05]. For presentation simplicity, we use "range query" to denote "probabilistic threshold range query" whenever there is no ambiguity.

Definition 7.1. ***Probabilistic Threshold Range Query***
For a given set of uncertain objects $\mathcal{U}$ and a range query Q, the probabilistic threshold range query retrieves all uncertain objects $U \in \mathcal{U}$ with $P_{app}(U, Q) \geq \theta$ where θ is the user specified probabilistic threshold and $0 < \theta \leq 1$.

In some applications it suffices to get an approximate number of uncertain objects instead of retrieving the uncertain objects qualifying the range query. We call this size estimation of range query.

Definition 7.2. ***Size Estimation of Range Query***
For a given set of uncertain objects $\mathcal{U}$ and a range query Q, estimate the number of uncertain objects $U \in \mathcal{U}$ with $P_{app}(U, Q) \geq \theta$ where θ is the user specified probabilistic threshold and $0 < \theta \leq 1$.

To handle results with low appearance probability $P_{app}(U, Q)$, ranking the objects based on $P_{app}(U, Q)$ and returning top-k results only is another method besides probabilistic threshold based approach. Following is the problem definition.

Definition 7.3. ***Top-k Range Query***
For a given set of uncertain objects $\mathcal{U}$ and a range query Q, a top-k range query retrieves k uncertain objects $U \in \mathcal{U}$ with highest $P_{app}(U, Q)$.

Efficient processing of joins often relies on fast execution of range query in the filtering phase. The problem of distance based spatial similarity join over uncertain data is introduced in [KKPR06]. Following is a formal definition of this problem in a probabilistic threshold fashion. It is referred as "similarity join" when there is no ambiguity.

Definition 7.4. ***Probabilistic Threshold Similarity Join***
For two given sets of uncertain objects $\mathcal{U}$ and $\mathcal{V}$, retrieve all pairs of (U, V) where $U \in \mathcal{U}$ and $V \in \mathcal{V}$ such that $\int_{x \in U_r} \int_{y \in V_r \wedge |x-y| \leq \gamma} U.pdf(x) \times V.pdf(y) dy dx \geq \theta$. γ and θ are pre-defined distance and probabilistic threshold respectively.

Problem Statement

In this chapter, we aim to build an efficient index to support various queries which rely on efficient processing of range query. The index supports uncertain objects with arbitrary PDFs and is not sensitive to the size and shape of the query regions.

7.1.2 Preliminaries

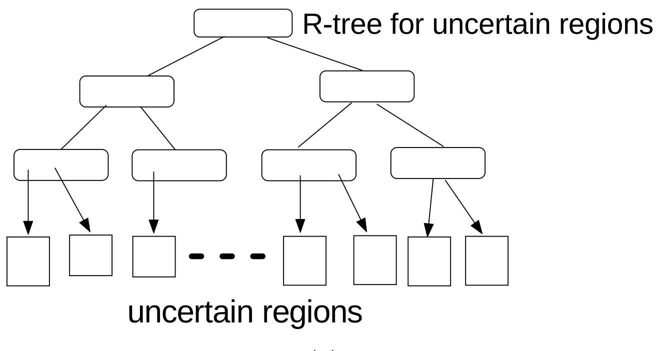
In this subsection, we first briefly describe and analyse two types of indexing structures supporting uncertain objects with arbitrary PDFs, *R-Tree based index* and *PCR based index*. Then we introduce the *inverted index* technique and its application in indexing uncertain objects. In the end is a brief introduction of other existing techniques.

R-Tree based Index

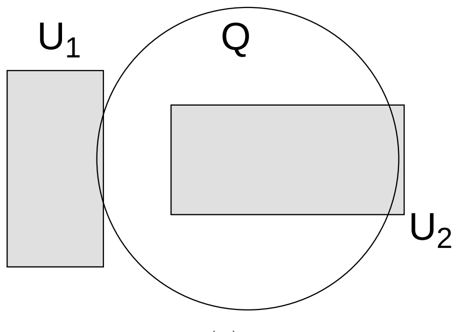
R-Tree family[Gut84] are tree data structures which are similar to B-Tree, but are used for spatial access methods in which a set of points or rectangles are recursively grouped. Each intermediate entry of R-Tree is represented as a MBR, which is the *minimal bounding rectangle* of the entry which tightly bounds all the data in the subtree. R-Tree can efficiently support the range query because it can prune or validate a group of objects at intermediate entries. Moreover the construction of

R-Tree aims to maximise the chance of pruning/validating R-Tree entries for the range query as well.

A simple way to index the uncertain objects is to organize their uncertain regions with existing indexing approaches like R-Tree [KKPR06, KHKH07, SMP$^+$07, CKP04, LBR$^+$07]. Figure 7.2(a) illustrates the basic idea of the uncertain region based indexing where the uncertain regions of the uncertain objects are indexed by R-Tree. It is simple and performs well if the uncertain regions of objects are very small regarding the query region size. As the uncertain region is considered as an atomic unit in the index, without further exploring the detailed information it can not tell whether or not an uncertain object satisfies the query when uncertain region overlaps range query. Such an index inherently limits the filtering capacity of the index and is not suitable to the probabilistic threshold related queries. As shown in Figure 7.2(b), for a given query Q and probabilistic threshold $\theta = 0.5$, we can not prune U_1 although intuitively the $P_{app}(U_1, Q)$ should be small. Similarly, U_2 can not be validated either. Consequently, the performance of the index is poor when the size of the uncertain region is not small.



(a)



(b)

Figure 7.2: Uncertain Region Based Index

PCR based Index

PCR (*probabilistically constrained regions*) based indexes make use of the detailed information about PDF of uncertain objects to enhance the filtering capacity. It is introduced by Tao *et al* [TCX+05] to support the range query on uncertain objects in a multi-dimensional space where the PDF of the uncertain object might be arbitrary functions. PCR is a general version of *x-bounds* which aims to index one dimensional uncertain data [CXP+04].

In [TCX+05], an uncertain object U is modeled by its PDF $U.pdf(x)$ and uncertain region U_r. For a given probabilistic threshold θ, corresponding $U.pcr(\theta)$ can be employed for pruning and validating purpose. $U.pcr(\theta)$ is constructed as follows. As shown in Figure 7.3, in each dimension, two lines are calculated. In the horizontal dimension, U has the probability θ to occur on the left side of line l_{1-}, also probability θ to occur on the right side of line l_{1+}. Similarly, l_{2-} and l_{2+} are calculated in the vertical dimension.

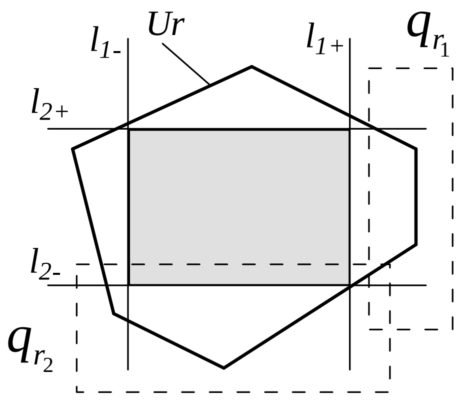


Figure 7.3: A 2d PCR (θ)

The shadowed region in Figure 7.3 forms $U.pcr(\theta)$. A series of theorems are proposed to take advantage of $U.pcr(\theta)$ to prune or validate U regarding θ. As shown in Figure 7.3, suppose both range queries q_1 and q_2 have the same probability threshold $\theta \leq 0.5$ and with query regions q_{r_1} and q_{r_2} respectively. U can be pruned regarding q_1 because q_{r_1} does not intersect with $U.pcr(\theta)$. On the other hand, U can be validated with respect to q_2 since q_{r_2} completely contains U_r below l_{2-}. As it is infeasible to keep all $U.pcr(\theta)$ for any $\theta \in (0, 1]$, a finite number of PCRs are precomputed to facilitate the range query process in [TCX+05]. Based on the PCRs of the uncertain objects, U-Tree is built up in a similar way with R-Tree where each entry in a leaf node corresponds to an uncertain object. The main difference is the splitting process in which U-Tree focuses on optimizing the filtering capacity of the PCRs in the intermediate node.

In order to prune or validate an uncertain object U-Tree needs to project the query region to each dimension as shown in Figure 7.4. This loses the spatial "clustering" information which in turn might severely impair the filtering capacity of the PCR technique. As shown in Figure 7.4, if the query region Q_r is a rectangle which does not align the x and y axis, then there is no difference between Q_r and M (shadowed rectangle) regarding the pruning ability to uncertain object U_1. It implies that we can not prune U_1 for query Q regardless of the probabilistic threshold value, even though they do not intersect with each other at all. Query Q in Figure 7.4 is not uncommon in real applications. For instance, it could be a buffer query [Sad05] based on a segment of roads or rivers, which is a popular query in many Geographic Information System(GIS) applications [Sad05]. Another case is illustrated in Figure 7.5 where the range query is a circle. As suggested in [TCX+05], two rectangles R_1 and R_2 are utilized for pruning and validation respectively. This inherently weakens the filtering capacity of U-Tree. As in Figure 7.5, U-Tree loses its pruning capacity in the striped areas. As we know, the range query with a circle region is very popular in distance based queries. Moreover, it is essential for the spatial similarity joins.

Inverted Index

In information technology, an inverted index maps from content, such as a word, to its locations in a database file or a document to support full text search. Each

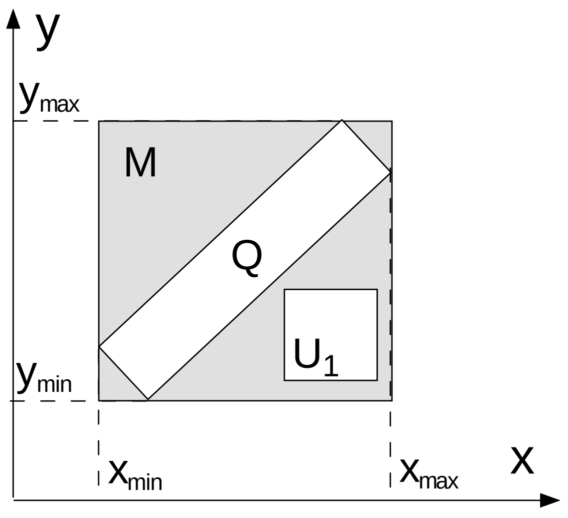


Figure 7.4: Irregular Query

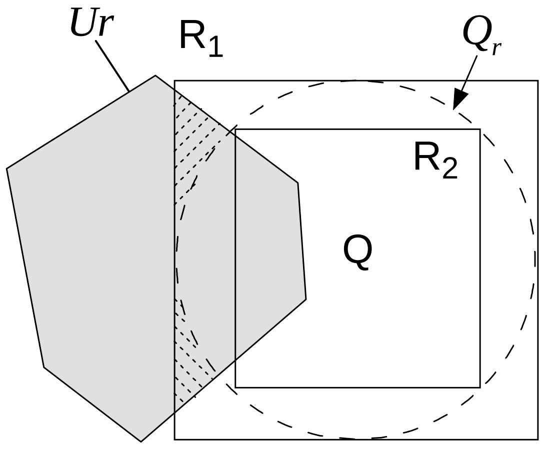


Figure 7.5: Circle query

words is allocated with a set of posting entries *(docID, offset/frequency)* which are sorted by the offset or frequency. Inverted index techniques are employed in [AY08, MKM08, SMP$^+$07] for indexing uncertain objects in specific applications with assumption or constraints on objects' PDFs or types. The R-Tree and inverted index techniques are employed in the problem of keyword searching on spatial database [HHLM07, FHR08] as well in which the keyword occurrence and the document location are considered. The problem they studied is inherently different with ours.

There are also some studies on indexing uncertain objects which focus on special cases of objects' PDF or particular data types. For instances, in [BGK$^+$07, BPS06b], Böhm *et al* study range queries with the constraint that PDF of uncertain objects is *Gaussian* distribution. Managing uncertain trajectories [Din08], existentially uncertain data [DYM$^+$05], uncertain categorical data [SMP$^+$07], vague spatial objects [ZBG07] have been separately studied. Aggarwal *et al* [AY08] study the problem of indexing high dimensional uncertain data with the assumption that the PDF of the uncertain object on each dimension is independent with others.

An index structure called *UniGrid* is proposed to efficiently support the similarity and range query on a selected subset of dimensions. In [MKM08], Ma *et al* propose solutions for efficient retrieval of uncertain spatial point data where the location information is derived from the free text by *spatial expressions*. With an assumption that the space is partitioned by a *virtual grid* with limited number of cells and a region (region of an uncertain object and region of the query) either occupies a whole cell or does not intersect with it at all, a grid index named *U-grid* is built for efficient spatial query processing.

Motivated by the above analysis of existing indexing techniques, we aim to develop a partition based index structure such that the spatial "clustering" information can be kept and the filtering capacity is less sensitive to the shape of query region. Moreover the structure should be space efficient and support arbitrary PDF.

7.2 UI-Tree Index

Based on the analysis of existing index structures for multidimensional uncertain objects, we develop an R-Tree based inverted index technique which is based on the partitions of uncertain objects. Section 7.2.1 introduces the motivation of our index structure and some important index building criterions. Then we describe the details of the index structure and its maintenance algorithms in Section 7.2.2 and Section 7.2.3 respectively.

7.2.1 Index Building Criterions

As discussed in Section 7.1, since R-Tree based techniques do not capture any details of the PDF of uncertain objects, the performance is poor when the size of the

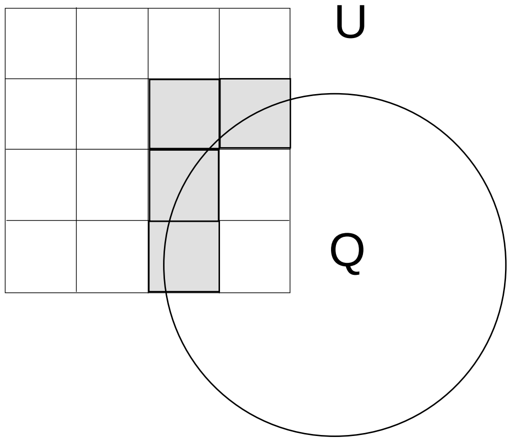


Figure 7.6: Motivation

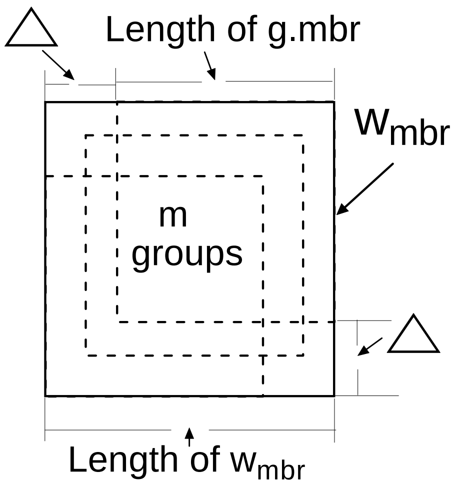


Figure 7.7: Size of w_{mbr}

uncertain region is not very small. Although the PCR based technique makes use of the PDF information by pre-computing the *probabilistically constrained regions*, the spatial "clustering" information of the instances is lost because the computation is based on projection on each dimension. So it is sensitive to the shape of the queries. Based on these observations, instead of building index structure against the uncertain region or PCR, we construct the index based on the partitions of the uncertain objects such that the spatial "clustering" information of instances of an uncertain object is well preserved. Following is the motivation of our UI-Tree technique. Note that our analysis focuses on the range query as it is fundamental to other queries studied in the chapter.

Suppose we partition each uncertain object into l disjointed groups $\{g_i\}$ such that for any instance $x \in U_r$, x is contained by one and only one group. And each group g_i consists of the object identity, probability and minimal bounding rectangle (MBR) of the group which are denoted by $g.oid$, $g.p$ and $g.mbr$. Note that the probability of the group is the accumulation of the probability of all instances within that group. The advantage of the partition is immediate. As shown in Figure 7.6, suppose the uncertain object U is partitioned into 16 groups $\{g_1, g_2, \ldots, g_{16}\}$ and

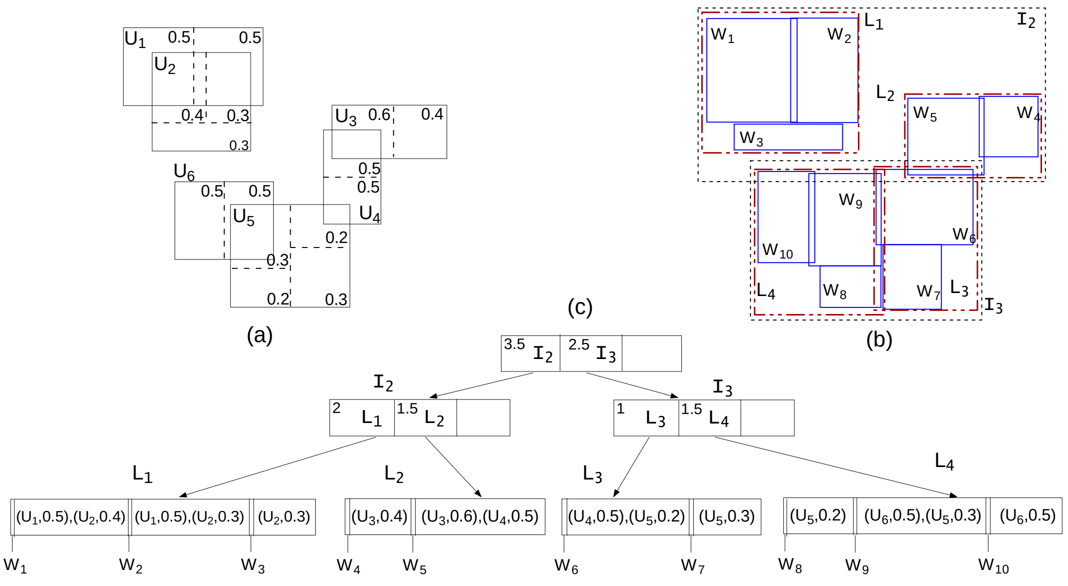

(a)
(b)
(c)
3.5 I2 | 2.5 I3
2 L1 | 1.5 L2
1 L3 | 1.5 L4
(U1,0.5),(U2,0.4) | (U1,0.5),(U2,0.3) | (U2,0.3)
(U3,0.4) | (U3,0.6),(U4,0.5)
(U4,0.5),(U5,0.2) | (U5,0.3)
(U5,0.2) | (U6,0.5),(U5,0.3) | (U6,0.5)
W1 W2 W3 W4 W5 W6 W7 W8 W9 W10

Figure 7.8: R-Tree Based Inverted Index

each group has probability $\frac{1}{16}$. Let $LP_{app}(U,Q)$ and $UP_{app}(U,Q)$ denote the lower and upper bound of the appearance probability of uncertain object U regarding query Q, then we have $LP_{app}(U,Q) = \frac{2}{16}$ and $UP_{app}(U,Q) = \frac{6}{16}$.

Clearly, the larger the number of partitions, the better filtering capacity since the gap between $LP_{app}(U,Q)$ and $UP_{app}(U,Q)$ comes from the accumulated probabilities of the groups overlapping with query region Q_r. However, in order to construct an index with decent filtering capacity we need to partition each uncertain objects into a certain number of groups. This number might be large and hence prevent the use of this approach in the applications with a large number of uncertain objects. So we consider to merge some groups of uncertain objects such that the index size can be reduced. Following is the first criteria for our index structure:

Index Building Criteria 1. In order to control the size of the index, we need to develop algorithm to merge the groups of partitions from uncertain objects.

As shown in Figure 7.8(a), suppose six uncertain objects U_1, U_2, $\ldots$, U_6 are partitioned into groups by the dashed lines. Figure 7.8(b) illustrates the merge result of these groups, denoted by w_1, w_2, $\ldots$, w_{10}. We call w_i a "word" in the spatial space. Suppose w is the *word* constructed from a set of groups $\{g_i\}$, then w_{mbr} is the minimal bounding rectangle of all $g_i.mbr$s. Clearly, we prefer a w_{mbr} with small size. Figure 7.8(b) illustrates the w_{mbr} created by merging MBRs of m groups. Assume dimensionality of the space is d and the domain of each dimension is normalized to $[0, 1]$ and let g_j and w_j denote the average length of $g.mbr$ and w_{mbr} at j-th dimension where $1 \leq j \leq d$. According to the analysis in [TS96], if there are totally $l \times n$ groups whose locations are *independent* with each other and every m of them whose MBRs are close to each other are merged to form k *words* where $k = \frac{l \times n}{m}$, we have $w_j = g_j + \bigtriangleup$ where

$$\bigtriangleup = \frac{m^{\frac{1}{d}} - 1}{(n \times l)^{\frac{1}{d}}} \tag{7.1}$$

For the given m and l, Formula 7.1 implies the average length of w_{mbr} on each dimension decreases with the number of uncertain objects which is confirmed in our experiment.

In order to distinguish groups from different uncertain objects and their probability values during the query processing, we have the second index building criteria:

Index Building Criteria 2. The object identity and probability values for each group should be kept after it is merged.

The *inverted index* technique is employed to meet this criteria. For each word w, a posting list denoted by w_{list} is maintained to keep the object identity and probability of groups merged to w. w_{list} consists of set of tuples $t(oid, p)$ where oid and p are the object identity of the group and its probability. The tuples in w_{list} are sorted decreasingly by the probability values. And w_p is the total probability of the tuples in w_{list}.

We use $t_{w,U}$ to denote a tuple which belongs to the posting list of *word* w with object identity U. $P(t_{w,U})$ denotes the probability of $t_{w,U}$. Note that probability values of groups from the same uncertain object will be accumulated in the list. Suppose we construct an index structure based on a set of words $\{w_i\}$ such that each group of the uncertain objects belongs to one and only one *word*. For uncertain object U, $\mathcal{W}(U)$ represents the set of *words* such that for any $w \in \mathcal{W}(U)$, there is a posting tuple $t_{w,U}$ in w_{list}. For presentation simplicity, we say $\mathcal{W}(U)$ is the *words* contained by U. And if U does not contain w, we have $P(t_{w,U}) = 0$. Let $\mathcal{W}$ and $|\mathcal{W}|$ denote all words in the space and its size. A small $|\mathcal{W}|$ implies that more groups are merged and therefore a small index size. On the other hand, the size of the posting list for each *word* increases with the number of partitions for each uncertain object. Therefore, we can control the size of the index by $|\mathcal{W}|$ and number of partitions for each uncertain object.

Theorem 7.1 indicates that we can compute the appearance probability bounds based on the inverted index of $\mathcal{W}$.

Theorem 7.1. *For a given query Q and an uncertain object U, let $\mathcal{W}_{con}$ ($\mathcal{W}_{over}$) denote the words in $\mathcal{W}(U)$ whose MBRs are contained (overlapped) by Q_r ($\mathcal{W}_{con} \cap \mathcal{W}_{over} = \emptyset$). We have*

$$\begin{aligned} LP_{app}(U,Q) &= \sum P(t_{w,U}), \ \textit{where } w \in \mathcal{W}_{con} \\ UP_{app}(U,Q) &= \sum P(t_{w,U}), \ \textit{where } w \in \mathcal{W}_{over} \cup \mathcal{W}_{con} \end{aligned}$$

Proof. For any instance $x \in U_r$ but $x \notin Q_r \cap U_r$, suppose x is allocated to group g in the partition of uncertain object U. Let w denote the *word* g belonging to, it is immediate that $w \notin \mathcal{W}_{con}$ which implies that there is no instance $x \notin Q_r \cap U_r$ contributes to the lower bound computation. Similarly, for any instance $x \in Q_r \cap U_r$, it will contribute to the upper bound computation. So the correctness of the lemma follows. □

Theorem 7.1 implies that the tightness of the bounds is affected by the *words* which overlap with query region Q_r. The probability of a *word* overlaps query region depends on the area of the MBR of w, denoted by $A(w)$, and the contribution to uncertainty is related to w_p which is accumulated probabilities of all tuples in its posting list. So we need to consider not only the area of MBRs of each *word* but also its probability value. For each *word* w, we use $WA(w)$ to represent $A(w) \times w_p$, called "weighted area" of the *word*. The third index building criteria is as follows:

Index Building Criteria 3. For the effectiveness of the index, given the number of *words* k, we want to create k *words* for the partitioned groups of uncertain objects such that $\sum_{1 \leq i \leq k} WA(w_i)$ is minimized.

Take the probabilistic threshold range query as an example, we further explain the intuition of this criteria based on following Lemma.

Lemma 7.1. *Suppose $\mathcal{W}$ is constructed for uncertain object set $\mathcal{U}$. Then for a range query Q, we assume the probabilistic threshold θ is randomly chosen from* $(0, 1]$ *and the probability of each* word *overlapping with Q_r is independent with each other, which is denoted by $P_{over}(w)$. Then the expected size of candidate set C is as follows if Theorem 7.1 is applied for appearance probability computation.*

$$|C| \;=\; \sum_{w \in \mathcal{W}} P_{over}(w) \times w_p$$

Proof. Let $\mathcal{W}_{over}(U, Q)$ denote the set $\{w\}$ such that w_{mbr} overlaps Q_r and $t_{w,U} \in w_{list}$. Given Q and $\mathcal{W}_{over}(U, Q)$, according to Theorem 7.1 $UP_{app}(U) = LP_{app}(U) + \sum P(t_{w,U})$ where $w \in \mathcal{W}_{over}(U, Q)$. Then U will be *validated* or *pruned* if

$LP_{app}(U) \geq \theta$ or $UP_{app}(U) < \theta$. As θ is randomly chosen from $(0, 1]$, $LP_{app}(U) \geq 0$ and $UP_{app}(U) \leq 1$, this implies that U will be kept in C with probability $\sum P(t_{w,U})$. So we have

$$\begin{aligned} |C| &= \sum_{w \in \mathcal{W}(U)} \sum_{U \in \mathcal{U}} P(t_{w,U}) \times P_{over}(w) \\ &= \sum_{w \in \mathcal{W}} P_{over}(w) \times w_p \end{aligned}$$

Recall that $P(t_{w,U}) = 0$ if U does not contain *word* w. □

As $P_{over}(w)$ depends on $A(w)$ and smaller $A(w)$ implies smaller chance to overlap with Q_r, we assume $P_{over}(w) = A(w) \times c$ where c is a constant derived from Q_r. Then it is immediate that we need to minimize $\sum_{1 \leq i \leq k} WA(w_i)$ for a small $|C|$ which is the measurement of filtering capacity of the index. Although the assumption of the independence of $P_{over}(w)$ among *words* and the existence of constant c is not practical in real world, this example does provide some insights for the index building criteria 3.

As a special case of the optimization problem in criteria 3 where all $w_i.p = 1$ and $k = 2$ is equivalent to the bipartition problem with measurement of area [HS89] which is NP-hard , we have to find some heuristics to solve this problem. Started with $n \times l$ *words* each of which corresponds to a group of the uncertain objects where n is the number of uncertain objects and l is the number of partitions for each uncertain object, we can create the index with a greedy heuristic such that the total "weighted area" is minimized at each step in which one *word* is merged. Nevertheless, this approach is infeasible to our problem as the computational complexity of the algorithms is cubic to $n \times l$ and multi-scan of the groups is required which leads to large number of IO operations.

In order to incrementally maintain the index structure in an efficient way, we employ R-Tree technique for the index construction because of its good support for spatial clustering [HK01]. Another important reason to apply R-Tree technique is because it can efficiently support a wide range of spatial queries. And this meets the fourth index building criteria.

Index Building Criteria 4. To efficiently support spatial queries which essentially depend on range query, the *words* should be well organized such that for the given query region Q_r, the *words* from $\mathcal{W}_{con}(Q)$ and $\mathcal{W}_{over}(Q)$ can be retrieved in an efficient way, where $\mathcal{W}_{con}(Q)$ ($\mathcal{W}_{over}(Q)$) denotes the $w \in \mathcal{W}$ which is contained(overlapped) by query region Q_r.

To address four index building criterions proposed, in the following part we introduce the R-Tree based inverted index technique for uncertain objects, named UI-Tree.

7.2.2 UI-Tree Structure

UI-Tree index is a depth-balanced tree structure similar to R-Tree [Gut84] as illustrated in Figure 7.8(c) and each node corresponds to a disk page. In the chapter, we use I , L and w to represent the non-leaf node, leaf node and *word* respectively. Each entry of the leaf node is a *word* with its posting list, represented by (w_{mbr},w_{list}). Note that, for space efficiency the w_p will be computed on the fly based on posting entries in w_{list}. A set of entries are organized by a leaf node and the minimal bounding rectangle of the leaf node tightly contains the MBRs of the *words*. Note that for the index maintenance efficiency, if a *word* w occupies more than one page due to the large size of w_{list}, we simply create a new *word* w' to take half of the posting entries. As we have to keep a certain number of *words* for a decent filtering capacity, usually the size of w is not large. Because we aim to minimize the sum of $WA(w)$, the total probability of *words* in child entries is kept for each node to facilitate the tree structure maintenance. The non-leaf node of the UI-Tree is exactly the same as that of R-Tree except the probability value is kept in its entry at parent node. Note that we do not keep any object identity information on the non-leaf node.

Suppose the average size of each *word* is s_w and then the average node capacity(fan-out) of the leaf node is $\lfloor \frac{PageSize}{s_w} \rfloor$, denoted by f_l. The node capacity of non-leaf node is denoted by f_i. And the height h of an UI-Tree with k *words* is as follows :

$$h = 2 + \lceil \log_{f_i} \frac{k}{f_i \times f_l} \rceil \tag{7.2}$$

If we regard the leaf node as a data entry, the UI-Tree corresponds to a R-Tree with $\frac{k}{f_l}$ data entries and an extra level for leaf nodes. Then the Formula 7.2 is immediate[FSR87].

7.2.3 Index Maintenance

In this section, we first introduce the UI-Tree structure maintenance algorithms including uncertain object partition, insertion and deletion.

Uncertain Object Partition

Before inserting an uncertain object into UI-Tree, we need to partition the uncertain object into l groups such that any instance $x \in U_r$ belongs to one and only one group. Ideally, we want to find l groups such that the sum of $A(g.mbr) \times g.p$ is minimized. As the partition is conducted on every uncertain objects, the partition algorithm must be very efficient in terms of CPU time and number of IO. If each uncertain object is already organized by some hierarchical tree structures such as R-Tree [Gut84] and $Quad$-Tree [FB], we can directly choose the intermediate node as the group since the instances of the uncertain object are naturally clustered.

Otherwise, we employ a partition approach similar with *kd*-Tree[Ben75]. Starting with one group which is the uncertain region of the uncertain object, we recursively partition the groups into two parts with the same probability value along a particular dimension chosen in a round robin order. Suppose the depth of the partition is d_p, then it comes up with $l = 2^{d_p}$ groups. For the *discrete* case, the partition procedure has time complexity of $O(d_p \times n_i)$ where n_i is the number of instances in the uncertain object. Recall that an instance of the uncertain object in *discrete* case corresponds to a possible occurrence of the uncertain object. This is because in each iteration we can first find the median value of a set of n elements on the selected dimension with time complexity $O(n)$ [CLRS01] and then separate the groups into two parts with one scan. As to the *continuous* case, we can find the median value based on the cumulative density functions(CDF) of the uncertain object and the partition cost is depended on CDF.

Insertion

The insert operation of UI-Tree is similar with R-Tree except that the probability value of the node is considered and we need to merge *words* to reduce the space. We can regard the node in the UI-Tree as a virtual *word* with empty posting list. Then we redefine the area of the node as its "weighted area", and all of the operations in R-Tree which are related with area computation is updated such as *choose Leaf* and *node splitting* in UI-Tree.

In order to incrementally maintain the UI-Tree with limited space, we need to merge *words*. Let $w = merge\ (w_1, w_2)$ be the merged *word* from w_1 and w_2. Note that w_{mbr} is the minimal bounding rectangle of $w_{1_{mbr}}$ and $w_{2_{mbr}}$, while w_{list} consists posting tuples of w_1 and w_2 in which tuples with the same object identity are merged. To measure the loss of information caused by merging two *words*, we define the *similarity* of two *words* w_1 and w_2 based on w:

$$sim(w_1, w_2) \quad = \quad \frac{1}{WA(\ w\ ) - WA(\ w_1\ ) - WA(\ w_2\ )} \tag{7.3}$$

Note that we have $sim(w_1, w_2) = \infty$ when $w_1 = w_2$.

Clearly, we prefer to merge *words* with high similarity according to our index construction criteria. For a given k which is the maximal number of *word* the UI-Tree will maintain, we first randomly choose $\frac{k}{l}$ uncertain objects and partition them into k groups to build up the UI-Tree. The merge operation is not considered at this stage so the procedure is the same as that of R-Tree except the "weighted area" is considered. After this, we start to control the number of *words* by merging similar *words*. Note that a group g from the partition of U can be regarded as a *word* with one posting entry. Algorithm 7.1 illustrates the details of the insertion algorithm. The flag $startmerge$ is set $false$ before the UI-Tree construction.

After choosing leaf node, the insertion procedure is simple if the merge stage does not start. Otherwise, we need to merge the most similar pair of *words* among *words* in L and g. Suppose the most similar pair of current words and their similarity is kept in each leaf node denoted by L_{w_1}, L_{w_2} and sim_L respectively. We will merge g with the most similar *word* in the leaf node if their similarity value is greater than sim_L. Otherwise, L_{w_1} and L_{w_2} are merged, and g is inserted as a new *word*. Note that the split might be invoked as well after merging two *words* since a new posting entry is created although the number of *words* remains the same. After insertion, we need to update the related information(e.g. MBR, probability) on leaf nodes and its parents nodes. Following the index example in Figure 7.8(c), Figure 7.9(a) demonstrates how the leaf node L_4 is updated after inserting a new uncertain object U_7.

For presentation simplicity, we use $m = \frac{l \times n}{k}$ to measure the *words* compression ration of UI-Tree, named *merge factor*. Since the splitting procedure is quite complicate, we omit this part. Then the cost for an uncertain object insertion is $O(l \times (h \times f_i \times d + f_l^2 \times d))$ in the worse case. As the leaf node choosing takes time $O(f_i \times d)$ to find most similar subnodes at each level and the worst time complexity between Line 5 and Line 14 is $O(f_l^2 \times d)$. And it takes time $O(f_i \times d)$ to adjust the nodes at each level.

Algorithm 7.1 Insertion(UI, U)

```
Input: $UI$ : the $UI$-Tree,
    $U$ : uncertain object to be inserted
1:  $startmerge := false$;
2:  $\mathcal{G} := l$ groups from partition of $U$;
3:  for each $g \in \mathcal{G}$ do
4:          choose Leaf node $L$ for $g$;
5:          if $startmerge = false$ then
6:                  $startmerge := true$;
7:          else
8:                  $w :=$ the $word$ most similar with $g$ in $L$;
9:                  if $sim(w,\ g) > sim_L$ then $w := merge(w, g)$;
10:                 else
11:                         $L_{w_1} := merge(L_{w_1},\ L_{w_2})$;
12:                         Insert $g$ to $L$ as a new $word$;
13:                         Update $sim_L, L_{w_1}, L_{w_2}$;
14:         Adjust the $UI$-Tree by propagating changes;
15: end for each
```

Remark 7. *Recall that if there is only one word in the leaf node and its size exceeds the page size due to the large w_{list}. As discussed in7.2.2, we simply create a new word w' by duplicating the w_{mbr} and putting half of posting entries from w_{list} to*

w'_{list}. *This rarely happens in our experiment as* m *is not large under our problem setting.*

Discussion 7.1. *Instead of specifying the* k *value, we can control the index size by a threshold* p *for the similarity between words in a similar manner with BIRCH [ZRL96]. Started with large* p *value, only pairs of words with similarity larger than* p *can be merged. Once the index size exceeds a given space budget, the index is rebuilt based on a new threshold with smaller value. We do not use this strategy because it is not clear how to choose the threshold and the cost of rebuilding might be expensive.*

Deletion

For uncertain object U to be deleted, we first descendantly find all *words* $\{w\}$ which include posting tuples $t_{w,U}$. Then all tuples are removed from their corresponding posting list. The *words* with empty posting list are removed from the UI-Tree which is same as R-Tree. Based on index example in Figure 7.8(c), Figure 7.9(b) show how the leaf node L_4 of the index is updated after deleting the uncertain object U_6. The delete operation is simple and efficient. However it suffers from its inability to perform adjustment of MBR of the *word* if some posting tuples are removed. Because the cost of "shrinking" the MBR of a *word* w is expensive as we have to reload the uncertain objects which contribute to the w_{list}. Consequently, the filtering capacity of the UI-Tree might degrade if there are frequent deletions. Nevertheless, the UI-Tree is efficient in many of the real applications in which there are no frequent updates.

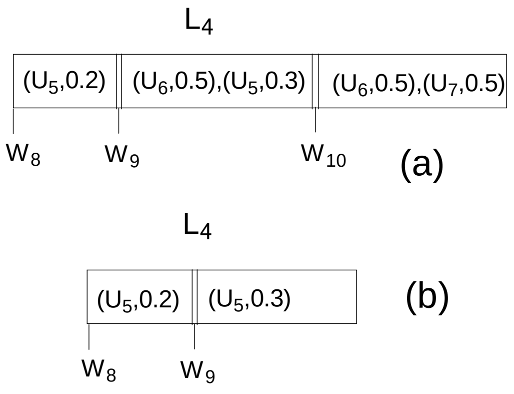


Figure 7.9: Update

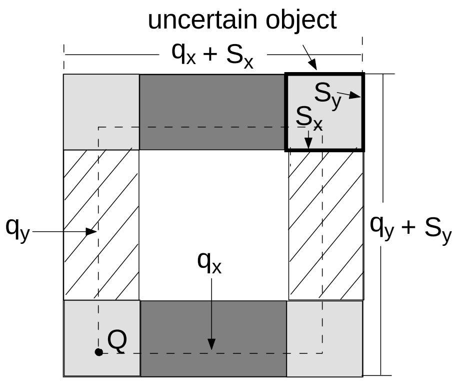


Figure 7.10: Example

7.3 Query Processing

In this section, we introduce how to efficiently process various queries based on the UI-Tree proposed in Section 7.2. Section 7.3.1 presents our range query algorithm and related analysis. Then we study the size estimation of range query in Section 7.3.2. Followed by top-k range query and similarity join in Section 7.3.3 and Section 7.3.4.

7.3.1 Range Query

In this subsection, we present a detailed searching algorithm for the probabilistic threshold range query based on the UI-Tree. For the given query Q, we descend the tree from the root in a manner similar to the R-Tree. All data entries(*words*) which are contained or overlapped by Q_r are retrieved, denoted by $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$ respectively. For each uncertain object U appeared in the posting lists of the *words*, we use the U_{low} and U_{upper} to represent the lower and upper bounds for the $P_{app}(U,Q)$ which can be computed based on $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$ according to Theorem 7.1. Then all uncertain objects which can not be filtered are kept in a candidate list C for *verification*. Algorithm 7.2 describes the range searching procedure. Input of the algorithm are UI : the UI-Tree over $\mathcal{U}$, Q : query with region Q_r, θ : Probabilistic threshold. Output of the algorithm are objects with $P_{app}(U, Q) \geq \theta$.

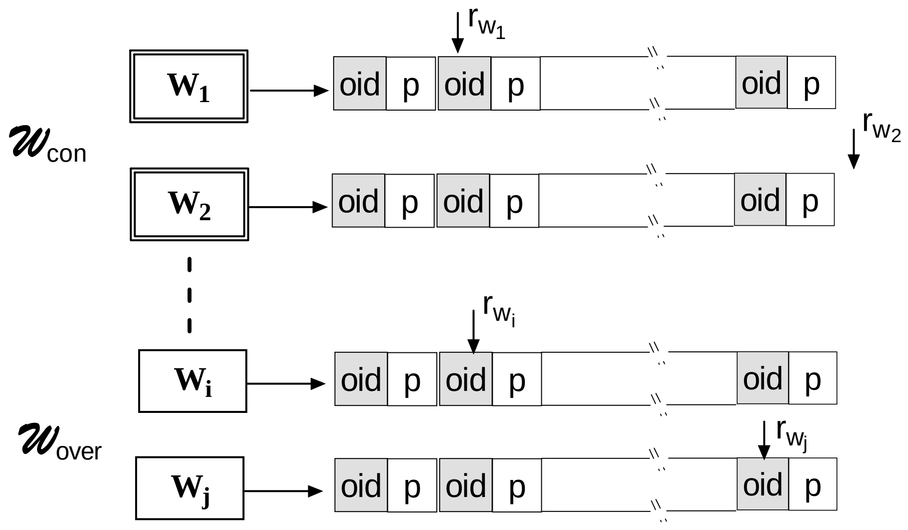


Figure 7.11: Query Example

The posting tuples of the list are visited in sequential order as shown in Figure 7.11. A pointer, denoted by r_w, is employed to record currently visited tuple in the posting list w_{list}. We refer a tuple as *current* tuple in the posting list if it is recorded by the pointer. Let P_{max} denote the total sum of probability values of all current tuples from $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. As the posting tuples are sorted decreasingly by their probability values, we can safely prune unseen uncertain objects once $P_{max} < \theta$. A maximal heap H is employed to maintain the pointers of the posting lists sorted by their probability values such that the P_{max} can be reduced in a greedy way.

The total cost of the range query is $C_R + C_{cand} + C_{ver}$. Specifically, C_R is the cost for retrieving leaf nodes containing *words* in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$ which is the same as the R-Tree range search. Let t_n denote the total number of posting tuples in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. The candidate set computation cost, denoted by C_{cand}, is $O(t_n \times \log w_n)$ in the worst case as the heap maintenance cost is $\log(w_n)$ for each iteration, where

w_n is the number of *words* in $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$. C_{ver} is the cost for *verification* including exact appearance probability computation and some extra *IO* cost for loading uncertain objects. Note that we do not discuss details of *verification* as the focus of the chapter is to develop efficient index technique to reduce the number of candidates of the queries.

Algorithm 7.2 Range Query(UI, Q, θ)

```
1: $\mathcal{L}$ := all leaf nodes in $UI$ contained or overlapped by $Q_r$;
2: $\mathcal{W}_{con}$ := *words* with $w_{mbr}$ contained by $Q_r$ from $\mathcal{L}$;
3: $C := \emptyset$; $R := \emptyset$; $H := \emptyset$; $P_{max} := 0$;
4: **for each** first posting tuple $t \in \mathcal{W}_{con} \cup \mathcal{W}_{over}$ **do**
5:     $r_w := t$; $P_{max} := P_{max} + t_p$; put $r_w$ into $H$;
6: **while** $H \neq \varnothing$ and $P_{max} \geq \theta$ **do**
7:     Remove top pointer $r_w$ from $H$; $t$ := the posting tuple $r_w$ referred;
8:     $U$ := the uncertain object with identity $t.oid$;
9:     **if** $U$ is not *pruned* or *validated***then**
10:        **if** $t$ is from $\mathcal{W}_{con}$ **then** $U_{low} := U_{low} + t_p$; $U_{upper} := U_{upper} + t_p$;
11:        **else** $U_{upper} := U_{upper} + t_p$;
12:        **if** $U_{upper} + P_{max} < \theta$ **then** $U$ is pruned;
13:        **else if** $U_{low} > \theta$ **then** $R := R \cup U$;
14:    **if** $t$ is not the last tuple **then** Let $r_w$ point to next tuple; put $r_w$ into $H$;
15:    update $P_{max}$;
16: Refine the $C$ by visiting remainding tuples;
17: **for each** $U \in C$ **do**
18:    **if** $P_{app}(U, Q) \geq \theta$ **then** $R := R \cup U$;
19: Return $R$;
```

Estimate the Filtering capacity

In the following part, with some uniformity assumptions we analyze the performance of the range query by estimating the number of candidates since it reflects the filtering capacity of the index. For presentation simplicity, we assume the domain sizes of all dimensions are between $[0, 1]$ and dimensionality is 2. Suppose the uncertain regions of the uncertain objects are regular rectangles whose instances follow the $uniform$ distribution and the probabilistic thresholds are randomly chosen from $(0, 1]$. As shown in Figure 7.10, we assume the query region is larger than the uncertain region of uncertain object on each dimension. Let s_x and s_y denote the average lengths of the rectangle on x and y dimensions respectively. A query Q has a regular rectangle region with length q_x and q_y, which is issued with a randomly selected centre.

Firstly, we assume there is no merge operation during the index construction $(m = 0)$. Let the probability $p_{cx} + p_{cy} + p_n$ represents the probability that Q_r overlaps U_r. Specifically, $p_{cx} = 2 \times (q_x - s_x) \times s_y$ is the probability of Q_r covering U_r at x dimension and $p_{cy} = 2 \times (q_y - s_y) \times s_x$ represents the probability of Q_r covering U_r at y dimension. While $p_n = 4 \times s_x \times s_y$ denotes the probability that Q_r overlaps U_r but does not cover U_r in any dimension. As shown in Figure 7.10, when the left-bottom corner of the query Q falls in the light grey rectangles with total area size $4 \times s_x \times s_y = p_n$, Q_r overlaps U_r but does not cover x or y dimension of U_r. Similarly, p_{cx} and p_{cy} correspond to the total area of dark grey rectangles and rectangles with strike lines.

Theorem 7.2 evaluates the expected candidate size for Q for $m = 0$.

Theorem 7.2. *Let C denote the set of uncertain objects in the candidate set in Algorithm 7.2* [2] *and suppose the uncertain region of each uncertain object is partitioned into $n_x \times n_y$ cells with same size, the average size of C can be estimated by $n \times (\frac{p_{cx}}{n_y} + \frac{p_{cy}}{n_x} + p_n \times \frac{(n_x+n_y)}{2 \times n_x \times n_y})$ where n is the number of uncertain objects.*

[2]For proof simplicity, we assume P_{max} is not considered in Algorithm 7.2

Proof 7.1. *According to the description of the Algorithm 7.2, an uncertain object U contributes to C if and only if U_r overlaps Q_r and $U_{low} < \theta \leq U_{upper}$. Let O_r denote the region such that Q_r overlaps U_r when the centre of Q, denoted by q_c, falls in O_r. Since probabilistic threshold θ is randomly chosen from $(0, 1]$, the probability of $U \in C$ is*

$$\begin{aligned} P_{U \in C} &= \int_{x \in O_r} pdf(x) \int_0^1 f(\theta, x) d\theta dx \\ &= \int_{x \in O_r} pdf(x) D(x) dx \end{aligned}$$

where $pdf(x)$ is the probabilistic density function of x. We have $f(\theta, x) = 0$ if $\theta > U_{upper}$ (being pruned) or $\theta \leq U_{low}$(being validated), otherwise $f(\theta, x) = 1$. We use $D(x)$ to represent the difference between U_{upper} and U_{low} when q_c locates at position x. According to Theorem 7.1, we have $D(x) = \sum_{w \in \mathcal{W}_{over}(U)} P(t_{w,U})$. For instance, $D(x)$ corresponds to the accumulated probability values of the shaded cells in Figure 7.12(a) and Figure 7.12(b).

As we assume the rectangle region of the query is larger than that of uncertain object in every dimension, following are three possible cases in which Q_r overlaps U_r:

$E1$: Q_r covers U_r in y dimension but not in x dimension.

$E2$: Q_r covers U_r in x dimension but not in y dimension.

$E3$: Q_r does not cover U_r in any dimension.

Case $E1$ is illustrated in Figure 7.12(a). According to the uniformity assumptions and Equation 7.4, we have

$$P_{E1} = p_{cy} \times \frac{1}{n_x}$$

where P_{E1} is probability of occurring of $E1$ and $U \in C$, p_{cy} is the occurrence probability of $E1$ and $p_{cy} = 2 \times (q_y - s_y) \times s_x$. Similarly, we have $P_{E2} = p_{cx} \times \frac{1}{n_y}$

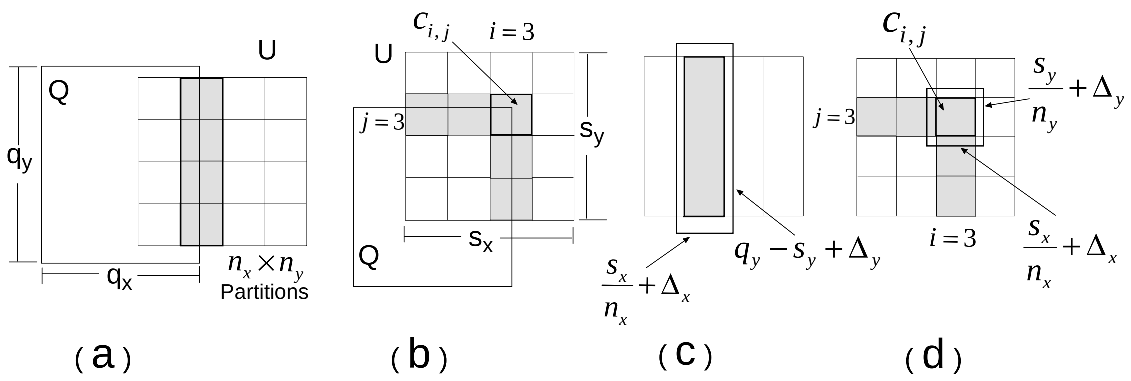


Figure 7.12: Filtering Capacity Evaluation

and $p_{cx} = 2 \times (q_x - s_x) \times s_y$. For case E3, Figure 7.12(b) illustrates one of its four sub-cases in which upper-right corner of the Q_r falls in U_r. Based on Equation 7.4 and uniformity assumptions, we have

$$\begin{aligned} P_{E3} &= 4 \times \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \frac{p_n}{4 \times n_x \times n_y} \times (i + j - 1) \times \frac{1}{n_x \times n_y} \\ &= p_n \times \frac{(n_x + n_y)}{2 \times n_x \times n_y} \end{aligned}$$

where p_n denotes the occurring probability of $E3$ which is $4 \times s_x \times s_y$. Since $P_{U \in C} = P_{E1} + P_{E2} + P_{E3}$, according to the uniformity assumptions the expected size of C is

$$E(C) = n \times P_{U \in C} + 0 \times P_{U \notin C} = n \times P_{U \in C}$$

where n is the number of uncertain objects.

Once the merge procedure is involved, let Δ_x and Δ_y denote the average increment of the MBRs of the partitions on x and y dimensions respectively. The following theorem evaluates the estimated candidate size for Q. Clearly, the more partitions of the uncertain objects are merged, the larger Δ values; And hence less filtering power.

Theorem 7.3. *Suppose a UI-Tree is constructed based on the partitions of the uncertain objects and the uncertain region of each uncertain object is partitioned* *into $n_x \times n_y$ cells with same size, then the expected size of C can be estimated by $n \times (\frac{p_{cx}}{n_y} + \frac{p_{cy}}{n_x} + p_n \times \frac{(n_x+n_y)}{2\times n_x \times n_y})$ where n is the number of uncertain objects, $p_{cx} = 2 \times (q_x - s_x + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y) \times n_y$, $p_{cy} = 2 \times (q_y - s_y + \Delta_y) \times (\frac{s_x}{n_x} + \Delta_x) \times n_x$ and $p_n = 4 \times (\frac{s_x}{n_x} + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y) \times n_x \times n_y$. Δ_x and Δ_y denote the average increment of MBRs of the partitions after merge procedure on x and y dimensions respectively.*

Proof 7.2. *As the MBRs of the partitions for the uncertain objects are merged during the index construction, the probability that Q_r overlaps those partitions increases. For instance, as shown in Figure 7.12(c), the probability that the left boundary of Q_r overlaps the i-th column in case $E1$ of Theorem 7.2 becomes $(q_y - s_y + \Delta_y) \times (s_x + \Delta_x)$. Similarly, Figure 7.12(d) illustrates that the upper-right corner of the query region falls in each partition with probability $(\frac{s_x}{n_x} + \Delta_x) \times (\frac{s_y}{n_y} + \Delta_y)$. Following the same rational of Theorem 7.2, the correctness of the Theorem is immediate.*

7.3.2 Size Estimation of Range Query

Instead of retrieving the uncertain objects qualifying the range query, it suffices to get the approximated number of uncertain objects in some applications. One of the important observations in the field of multi-dimensional selectivity estimation is that although the whole data set is unlikely to follow the uniform distribution in real applications, it might be true within a local area. This motivates us to estimate the selectivity of the range query based on the uniformity assumption of the instances in the MBRs of their corresponding *words*. Then instead of keeping lower and upper bounds, the appearance probability of the uncertain object U

regarding query Q can be estimated with following Formula:

$$\begin{aligned} P_{app}(U,Q) &= \sum_{w \in \mathcal{W}_{over}} P(t_{w,U}) \times \frac{A(Q_r \cap w_{mbr})}{A(w_{mbr})} \\ &+ \sum_{w \in \mathcal{W}_{con}} P(t_{w,U}) \end{aligned}$$

This implies that we can estimate the size of range query based on the UI-Tree only. Our experiments demonstrate the effectiveness of the estimation.

7.3.3 Top-k Range Query

For a given query Q, once we retrieve $\mathcal{W}_{con}$ and $\mathcal{W}_{over}$, the remaining part of the query processing is similar with the traditional top-k computation on distributive inverted indexes[FLN01]. The main difference is that the posting tuples from $\mathcal{W}_{over}$ only contribute to the appearance probability upper bound of uncertain object. In our implementation, we modify Algorithm 7.2 such that we can safely claim all of the unseen uncertain objects can not be top-k answers once $P_{max} \leq p_{lk}$ where p_{lk} is the k-th largest U_{low}. After the refinement, all uncertain objects with $U_{upper} < p_{lk}$ can be *pruned* and the ones with $U_{low} \geq p_{uk}$ are *validated* where p_{uk} is the k-th largest U_{upper}. Then we need to compute the exact $P_{app}(U,Q)$ for the uncertain objects in the candidate set to decide the top-k result.

7.3.4 Similarity Join

As the algorithm for probabilistic threshold similarity join is lengthy, we only introduce the outline of the algorithm. Let $\mathcal{U}$ and $\mathcal{V}$ denote two sets of uncertain objects. The distance and probabilistic threshold is represented by γ and θ respectively. Firstly, we retrieve all pairs of *words* w_s, w_r based on the traditional spatial join algorithm [BKS93] such that $|w_{s_{mbr}} - w_{r_{mbr}}|_{min} \leq \gamma$ where $|w_{s_{mbr}} - w_{r_{mbr}}|_{min}$ denote the minimal Euclidean distance between $w_{s_{mbr}}$ and $w_{r_{mbr}}$. These pairs are classified into two sets: $\mathcal{W}^*_{in}$ and $\mathcal{W}^*_{part}$. For any $(w_s, w_r) \in \mathcal{W}^*_{in}$, $|w_{s_{mbr}} - w_{r_{mbr}}|_{max} \leq \gamma$ which implies the distance between any pair of instances of uncertain objects in w_s and w_r is smaller or equal to γ. Other pairs belong to $\mathcal{W}^*_{part}$. Let $LP(U \bowtie_\gamma V)$ and $UP(U \bowtie_\gamma V)$ denote the lower and upper bounds for the similarity between uncertain objects U and V. Clearly, pairs from $\mathcal{W}^*_{in}$ contribute to the computation of $LP(U \bowtie_\gamma V)$ and $UP(U \bowtie_\gamma V)$, while the ones from $\mathcal{W}^*_{part}$ only contribute to $UP(U \bowtie_\gamma V)$. Similar to Theorem 7.1, we have following Formulas for *verification* and *pruning* respectively:

$$\begin{aligned} LP(U \bowtie_\gamma V) &= \sum_{(w_s,w_r)\in\mathcal{W}^*_{in}} P(t_{w_s,U}) \times P(t_{w_r,V}) \\ UP(U \bowtie_\gamma V) &= \sum_{(w_s,w_r)\in\mathcal{W}^*_{in}\cup W^*_{part}} P(t_{w_s,U}) \times P(t_{w_r,V}) \end{aligned}$$

The correctness of the Formulas is immediate based on the same rationale of Theorem 7.1. To compute the candidate efficiently, a pointer is employed for each posting list in a similar manner with Algorithm 7.2. All of the pairs are organized by a maximal heap sorted by the probability values of their corresponding posting entries. Let PS and PR denote the pointers in $\mathcal{U}$ and $\mathcal{V}$ respectively. Let PS_{max} and PR_{max} keep the maximal possible probability for unseen uncertain objects from $\mathcal{U}$ and $\mathcal{V}$ respectively. Clearly, once $PS_{max} \times PR_{max} < \theta$, the pair (U, V) does not belong to the candidate set if U and V are unvisited.

7.4 Performance Evaluation

We present results of a comprehensive performance study to evaluate the efficiency and scalability of proposed techniques in the chapter. Following algorithms are evaluated.

***UI*-Tree** The *R*-Tree based inverted index technique proposed in Section 7.2

and four query algorithms presented in Section 7.3.

U-Tree The U-Tree technique presented in [TCX+05]. The implementation is public available.

R-Tree The uncertain region based R-Tree technique. The implementation of similarity join is based on the join strategy proposed in [KKPR06]. As there is no existing work on the size estimation of range query and top-k range query on uncertain objects with arbitrary PDF, the R-Tree technique is also employed as baseline algorithm because it can be regarded as a special case of UI-Tree in which every uncertain objects is a *word*.

In our experiment, the uncertain region of the uncertain object is a circle or sphere with radius r_u varying from 50 to 500 with default value 100. Suppose the PDF of an uncertain object is described by 400 instances (*discrete* case) which follow two popular distributions *Normal* and *Uniform*. The *Normal* serves as default distribution with standard deviation $\frac{r_u}{2}$. Specifically, we use the *constrained normal* distribution such that the possible location of the instances are restricted in the uncertain region. Instances might be loaded into memory once an uncertain objects is required for *verification*. For *verification* efficiency, same as [BSI08], the instances of an uncertain object are organized by an aggregate R-Tree where aggregate value of a R-Tree node is the accumulation of the probabilities of its child instances. Note that each instance corresponds to one bin in [BSI08] since we consider the *discrete* case in the experiment.

Two real spatial datasets, *CA* and *US*, are employed to represent the centre of the uncertain regions. They contain $62K$ and $200K$ 2-dimensional points representing locations in Los Angeles and the United States respectively[3]. We also generate

[3]Available at http://www.census.gov/geo/www/tiger/

synthetic dataset *3D* with dimensionality 3 and size $200K$, in which the centres and instances of uncertain objects are uniformly distributed. All dimensions are normalised to domain $[0, 10000]$ and CA with *constrained normal* distribution is employed as the default data set. To study the similarity join between two sets of uncertain objects, two synthetic data sets, named $2d$ $10K$ and $2d$ $1K$, are created in which the centres of uncertain objects follow *Uniform* distribution and the instances follow *constrained normal* distribution.

A workload for range query and its size estimation query consists of 200 queries in our experiment. Same as [TCX$^+$05], the region of a range query Q is a circle or sphere with radius r_q. r_q varies from 500 to 1500 with default value 1000. The centres of the queries are randomly chosen from the centres of the target uncertain objects. Note that the query regions in a workload share the same r_q. In order to avoid favouring particular θ value, we randomly choose the probabilistic threshold $\theta \in (0, 1]$ for each query. Instead of specifying the probabilistic threshold θ, the value k varying from 50 to 250 is used for the top-k query.

As all of the algorithms investigated in the chapter follow the *filtering and verification* frame work, the cost of the query is largely dependent on the candidate size as the *verification* is expensive in terms of IO and CPU time. So the average candidate size of the queries is employed as the most important performance measurement in our experiments. In addition, the average number of IO and false positives are recorded as well as the average query response time.

All algorithms proposed in this chapter are implemented in standard C++ with STL library support and compiled with GNU GCC. Experiments are run on a PC with Intel Xeon 2.40GHz dual CPU and 4G memory running Debian Linux. The disk page size is fixed to 4096 bytes. In order to achieve a good filtering capacity, the catalog size of U-Tree is set to 9 for CA and US, and 10 for $3D$ as suggested in[TCX+05].

Table 7.2 below lists parameters which may potentially have an impact on our performance study. In our experiments, all parameters use default values unless otherwise specified.

Notation	**Definition (Default Values)**
r_q	the radius of query (1000)
r_u	the radius of uncertain object region(100)
n	number of uncertain object
θ	probabilistic threshold ($\in (0, 1]$)
k	k value in top-k query
m	merge factor (12)
l	number of partitions(32)

Table 7.2: System Parameters

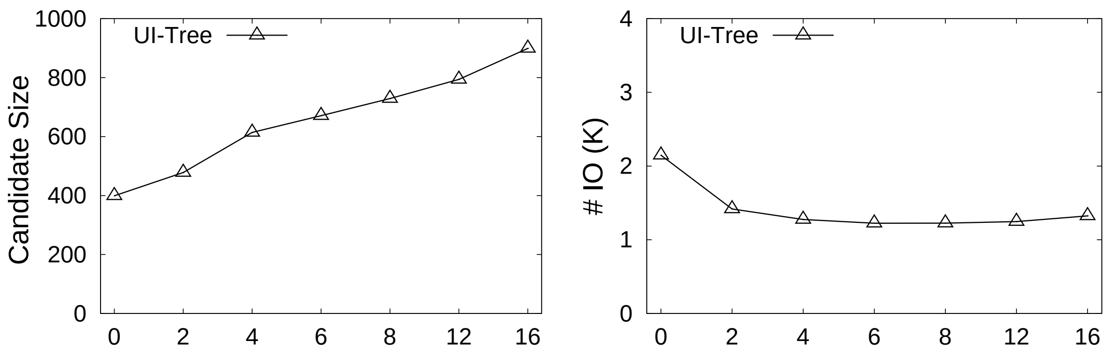
7.4.1 Index Construction Evaluation

In this section, we evaluate the performance of UI-Tree construction algorithm. The size of UI-Tree depends on the number of uncertain objects n, merge factor m and the number of partitions per uncertain object l. Clearly, smaller m and larger l will lead to better filtering capacity but more index space. For comparison convenience, we fix l to 32 and vary m to tune the index size such that it is similar with that of U-Tree. Under the default setting, Figure 7.13(a) shows the filtering capacity of the UI-Tree slowly decreases with m. Note that $m = 0$ implies there is no merge operation. Meanwhile the index size also drops from $62M$ to $15M$. Figure 7.13(b) reports the average IO cost for range query with default setting where m varies from 0 to 16. Although the number of candidate is small for small m, it might invoke more IO cost because of the large index size. By default, we set m to 12 for CA and US, and 6 for $3D$ respectively. So UI-Tree has similar index size with U-Tree.

Table 7.3 shows the index sizes of UI-Tree and U-Tree for CA, US and $3D$ respectively, which correspond to around 5%, 5% and 6% of the data sets.

	CA	US	$3D$
UI-Tree	15 (M)	49 (M)	80 (M)
U-Tree	15 (M)	49 (M)	86 (M)

Table 7.3: Index Size Comparison



(a) *Candidate Size vs m* (b) *IO* vs m

Figure 7.13: Index Evaluation against Diff. m

Note that since the U-Tree code from [TCX+05] does not employ the memory buffer during the U-Tree construction, the index construction is very slow. So we do not evaluate its construction time for fairness of comparison. Our index construction algorithm is very efficient, and the average insertion time per object for CA, US and $3D$ is around $1ms$, $4ms$ and $4ms$ respectively. More specifically, for CA, it totally takes $118s$ for partition, and $17s$($164s$) for insertion without(with) merge operation.

7.4.2 Query Performance Evaluation

In this section, we first evaluate the performance of range query. It is followed by size estimation, top-k range query and similarity join. As U-Tree is the state of the art technique for range query on multidimensional uncertain objects with arbitrary

PDF, it is employed as baseline algorithm to evaluate our UI-Tree based range query algorithm.

In order to confirm the observation of Figure 7.2(a) in Section 7.1.2, we construct a set of queries whose query regions are rectangles with length $2 \times r_q$ and width 300. The centre of the query is randomly chosen from CA data and we rotate the rectangle around its centre to a random angle between 0 and 2π. Figure 7.14 demonstrates that the filtering capacity of U-Tree degrades significantly with the growth of r_q from 500 to 1500, while the performance of UI-Tree technique is much more efficient and less sensitive to r_q.

To confirm the effectiveness of filtering capacity estimation, we also conduct filtering capacity evaluation on 2-dimensional synthetic data. There are $100K$ uncertain objects evenly distributed and the uncertain region of the uncertain objects are squares with width 200. The instances follow the *Uniform* distribution and query regions are regular rectangle queries with $q_x = q_y = 1600$. We build UI-Tree indices with m various from 0 to 16. Note that the candidate size estimation for $m = 0$ is based on Theorem 7.2 while others are based on Theorem 7.3 since the merge procedure is involved for $m > 0$. Figure 7.15 shows that estimated values are close to the real candidate size.

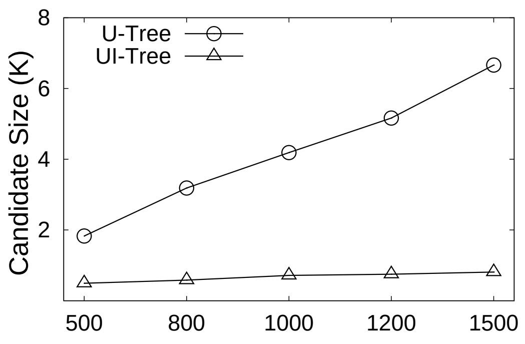


Figure 7.14: Diff. r_q

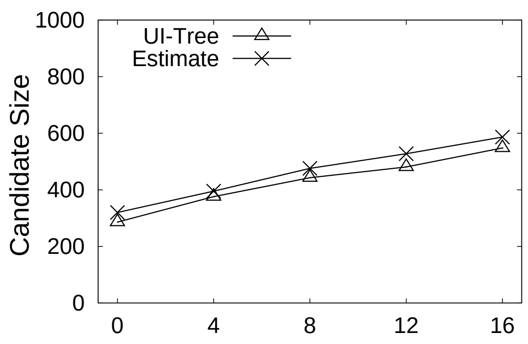
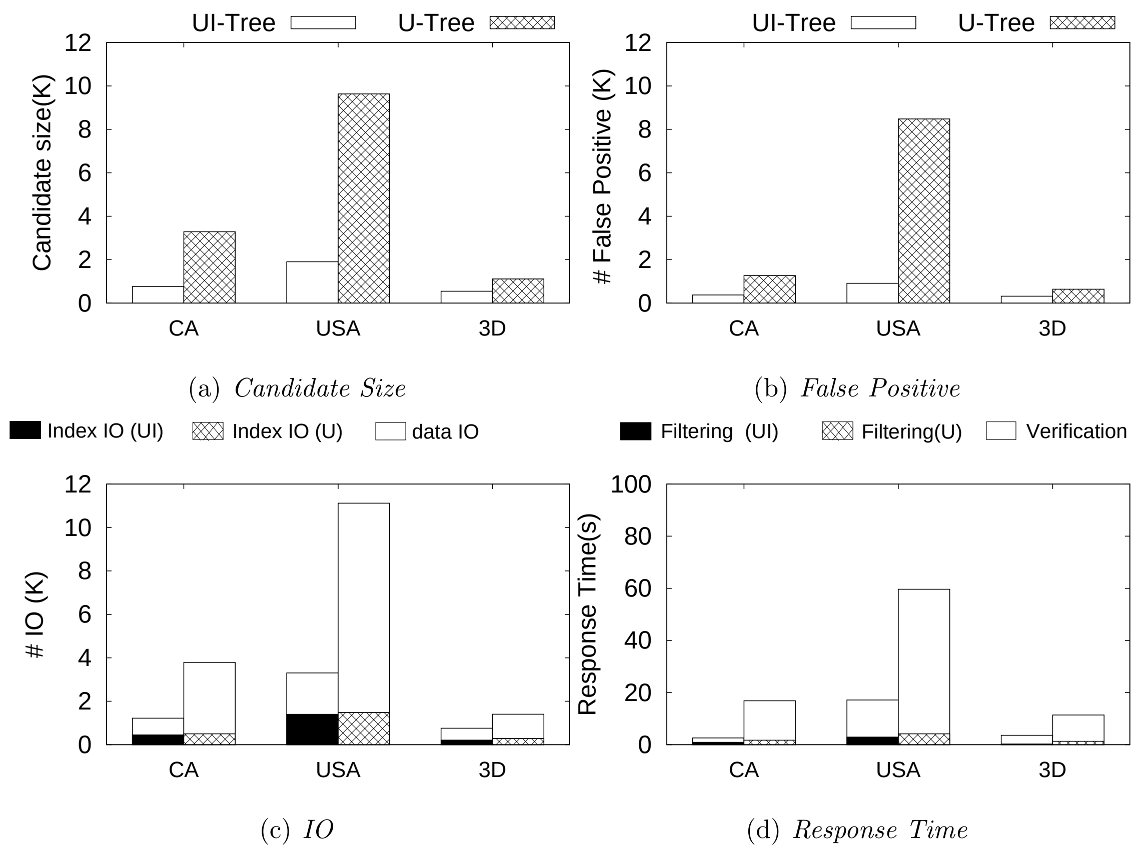


Figure 7.15: Diff. m

In the third set of experiments, we evaluate the performance of UI-Tree and

U-Tree against different dataset (CA, US and $3D$) while other system parameters are set to default values. The candidate size, number of false positive, number of IO and response time of two techniques are reported in Figure 7.16. Due to poor filtering capacity, the candidate size of U-Tree is much larger than that of UI-Tree especially on US as shown in Figure 7.16(a). A similar observation is found in Figure 7.16(b) which reports the number of false positives. According to Figure 7.16(c) and Figure 7.16(d), U-Tree and UI-Tree have similar filtering cost in terms of index IO and filtering time. However, due to the large number of candidates as shown in Figure 7.16(a), the total cost of the U-Tree for range query is much more expensive than that of UI-Tree in terms of IO and query response time.



(a) *Candidate Size* (b) *False Positive*

(c) *IO* (d) *Response Time*

Figure 7.16: Performance vs Diff. Data Set

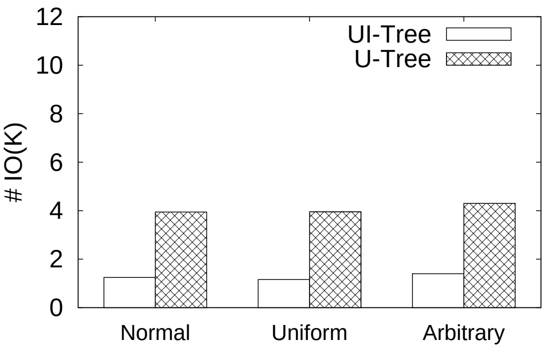


Figure 7.17: Diff. distribution

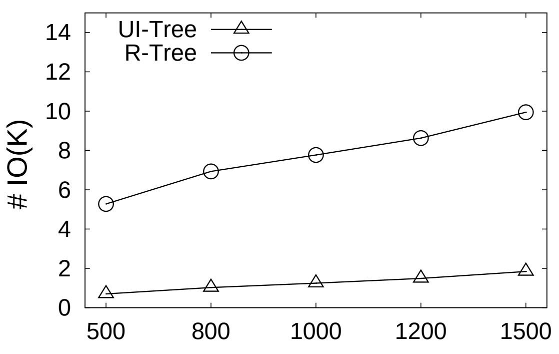


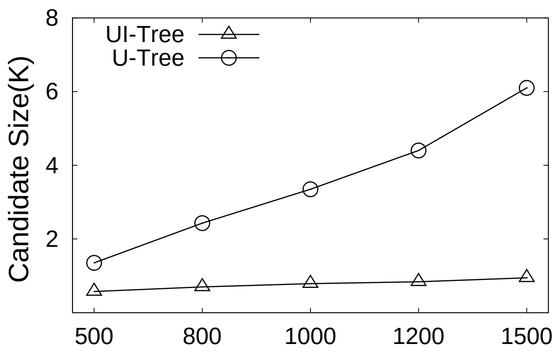
Figure 7.18: Diff. r_q

To evaluate the impact of the instance distribution of uncertain objects, we report the number of IO against different instance distributions. For the “arbitrary” distribution, the instances of the uncertain object are created by mapping 400 closest points for given random point in real data set US into the uncertain region of the uncertain object. Figure 7.17 shows the performance of the algorithm is not sensitive to these distributions.

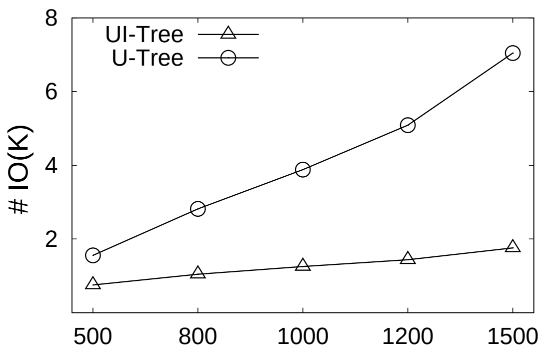
The R-Tree based index technique in [BSI08] can be used to support range query where the uncertain regions of the uncertain objects are indexed by a global R-Tree and the PDF of each uncertain object is represented by a set of bins(histograms) organized by an aggregate R-Tree. In our experiment, each instance corresponds to a bin as we consider the *discrete* case. Figure 7.18 demonstrates that UI-Tree significantly outperforms the R-Tree based index technique.

We evaluate the impact of r_q against the candidate size and the number of IO. Figure 7.19 shows that as r_q increases from 500 to 1500, the performance of U-Tree drops significantly while UI-Tree is more efficient and stable against r_q. In Figure 7.19(a), the candidate size of U-tree reaches $6K$ when $r_q = 1500$ which is 6 times larger than that of UI-Tree. And the same trend goes to the number of IO as depicted in Figure 7.19(b).

We study the scalability of U-Tree and UI-Tree by varying r_u and n. The results



(a) *Candidate Size vs* r_q



(b) *IO* vs r_q

Figure 7.19: Performance vs Diff. r_q Size

are reported in Figure 7.20 and Figure 7.21 respectively. As expected, the number of candidates of both techniques increases with r_u as the larger uncertain region implies more uncertain objects overlapping with query region. In Figure 7.21, a uniform sample of $50K$, $100K$ and $150K$ uncertain objects from US and US ($200K$) are employed to evaluate the impact of n. It is not surprising that the number of candidates goes up when the number of uncertain objects increases. Nevertheless, the growth of UI-Tree is much slower than that of U-Tree. This is because the filtering capacity of the PCRs of an uncertain object is independent with n. So the number of candidates goes linearly with n. According to the analysis in Section 7.2.1, for the fixed merge factor m, a larger n implies a smaller w_{mbr}. Consequently, the number of candidates grows very slowly with n because the filtering capacity of individual *word* improves due to a smaller w_{mbr}.

Figure 7.22 demonstrates that the probabilistic threshold does not have much impact on the candidate size of U-Tree and UI-Tree. And the performance of U-Tree slightly improves when θ is large but is still less competitive compared with that of UI-Tree.

In the last set of experiments, we evaluate the performance of UI-Tree based query algorithms for size estimation of range query, top-k range query and similarity

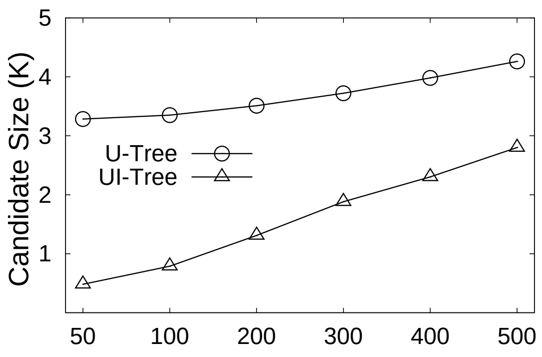


Figure 7.20: Diff. r_u

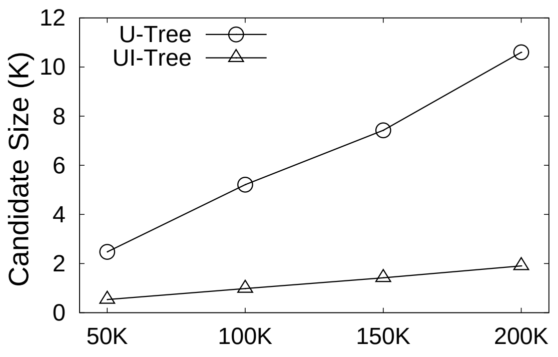


Figure 7.21: Diff. n

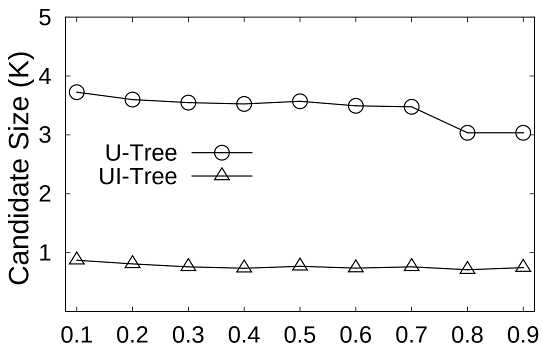


Figure 7.22: Diff. θ

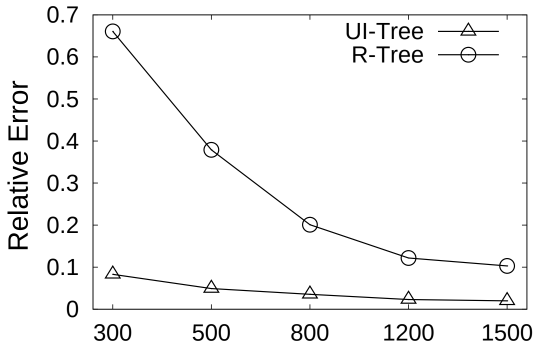


Figure 7.23: Size Est. vs r_q

join. The R-Tree based techniques are employed as baseline Algorithms. Let S and $\bar{S}$ denote the exact and estimated number of uncertain objects returned by range query respectively, we measure the effectiveness of the size estimation query by relative error $|\frac{S-\bar{S}}{S}|$. The average relative error of the queries under default setting is reported in Figure 7.23. As expected, the accuracy of UI-Tree technique is much better than that of R-Tree technique. It is 0.02 when r_q equals 1500.

We evaluate the number of candidates in top-k queries with k varying from 50 to 250. r_q is set to 300 and Figure 7.24 shows only a small number of candidates is required for further *verification* based on UI-Tree technique. There is no surprise to see the number of candidates is large and remains unchanged with k for R-Tree technique as it does not capture any details of the PDF of uncertain objects. Similar observation is reported in Figure 7.25 in which data set $2d$ $10K$ and $2d$ $1K$

are joined with different probabilistic threshold values varying from 0.1 to 0.9. The distance is set to 600.

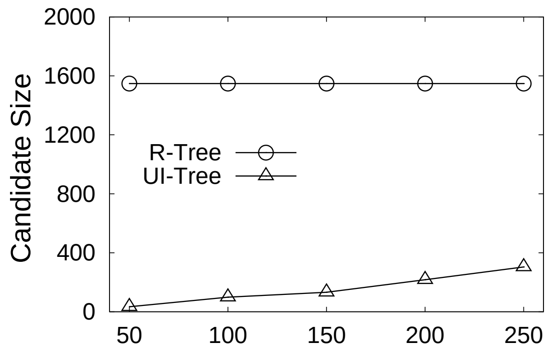


Figure 7.24: Top-k(Diff. k)

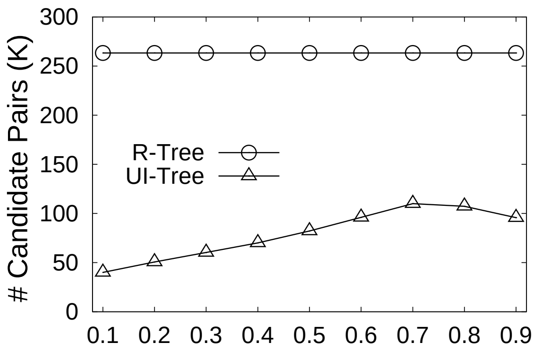


Figure 7.25: Similarity Join(Diff. θ)

7.5 Conclusion

In this chapter, we investigate the problem of effectively indexing multidimensional uncertain objects with arbitrary PDFs and propose a novel index structure UI-Tree-tree. Combining R-tree and inverted index techniques, UI-Tree-tree is space-effective and well supports different shapes of query regions. The dynamic maintenance of UI-Tree-tree is comprehensively studied. UI-Tree-tree serves as a general index where efficient processing of range queries is desirable. We propose solutions for various types of queries based on UI-Tree-tree, including range query, size estimation of range query, probabilistic top-k range query and similarity join. Our extensive experiments demonstrate that UI-Tree-tree significantly outperforms other state-of-the-art techniques.

Chapter 8

Conclusions and Future Work

In this chapter, we first conclude the thesis in Section 8.1. Then, Section 8.2 presents several directions of future work on developing advanced querying and indexing techniques on uncertain data.

8.1 Conclusions

It has been recognized that efficiently analyzing uncertain data is important for many applications including social networks, data cleaning, sensor data analysis, moving objects tracking, information retrieval, crime control, economic decision making, market surveillance, etc. Uncertainty is inherent in these applications due to various factors such as errors and imprecision in data, data randomness and incompleteness, limitation of measuring equipment, delay or lose of data updates and privacy preservation. Due to its importance, uncertain data analysis has recently attracted a great deal of attention from researchers. Numerous query types have been re-investigated under the uncertain semantics and many database management systems have been developed especially for uncertain data, as we summarized in Chapter 2.

In this thesis, we aim to bridge the gap of advanced query processing and indexing techniques on uncertain data. Particularly, we focus on the following four query types: 1) probabilistic top-k skyline query; 2) probabilistic skyline operator over sliding windows; 3) probabilistic threshold based top-k dominating query and 4) KNN search over multi-valued objects. Besides, we design an index structure UI-tree for uncertain data which overcomes the shortcomings of existing state-of-the-art techniques. Our main contributions are stated as follows.

Probabilistic Top-k Skyline Query We are the first to combine the feature of top-k objects with that of skyline to model the problem of top-k skyline objects against uncertain data. Efficient algorithms are proposed for both *continuous* and *discrete* cases.

Probabilistic Skyline Operator over Sliding Windows We are the first to study probabilistic skyline queries in the streaming environment. A candidate set with minimum size is characterized and efficient techniques are developed to answer the skyline queries continuously.

Probabilistic Threshold based Top-k Dominating Query We study the problem of efficiently computing top-k dominating queries on uncertain data. After formally defining the problem, we propose both exact and random algorithms to tackle the problem.

KNN Search over Multi-Valued Objects We propose to use quantiles to summarize relative-distribution-sensitive K nearest neighbors over multi-valued objects. Two different problem definitions are introduced and corresponding techniques are developed.

Effective Indexing Structure Noticing that existing index structures are sensitive to the size or shape of uncertain region and the queries, we introduce a novel R-Tree based inverted index structure, named UI-Tree, to efficiently support various queries, including range queries, similarity joins and their size estimation, as well as top-k range query, over multidimensional uncertain objects against continuous or discrete cases.

8.2 Future Work

8.2.1 Manipulating Complex Correlations

Currently, most existing query approaches are based on the assumption of simple correlations among uncertain data such as independence or mutual exclusiveness. Such assumptions usually fail to capture the uncertain semantics in real applications. A possible direction of future work is to manipulate complex correlations in probabilistic data. There are some key challenges. Firstly, correlations need to be derived and modeled from real applications. The correlations can sometimes be very complex to model and simplified models are necessary. Secondly, efficient and effective techniques need to be developed to process various queries on probabilistic data given the presence of complex correlations.

8.2.2 Building Prototype System

We plan to implement a prototype system to demonstrate the queries studied in this thesis. The dataset used will be the same as in the above three tasks. User friendly interface will be implemented, including dataset selection, query type selection and parameter setting.

Bibliography

[ABS$^+$06] Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Chris Hayworth, Shubha Nabar, Tomoe Sugihara, and Jennifer Widom. Trio: A system for data, uncertainty, and lineage. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1151–1154, 2006.

[AFS93] Rakesh Agrawal, Christos Faloutsos, and Arun N. Swami. Efficient similarity search in sequence databases. In *Proceedings of the Conference of Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[Agg08] Charu Aggarwal. On unifying privacy and uncertain data models. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, pages 386–395, 2008.

[AK08] Lyublena Antova and Christoph Koch. On APIs for probabilistic databases. In *Proceedings of the Management of Uncertain Data (MUD) Workshop*, pages 41–56, 2008.

[AKG87] Serge Abiteboul, Paris Kanellakis, and Gosta Grahne. On the representation and querying of sets of possible worlds. In *SIGMOD 1987*, pages 34–48, 1987.

[AKO07] Lyublena Antova, Christoph Koch, and Dan Olteanu. 10^{10^6} worlds and beyond: Efficient representation and processing of incomplete information. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 606–615, 2007.

[AS06] Serge Abiteboul and Pierre Senellart. Querying and updating probabilistic information in XML. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 1059–1068, 2006.

[AW07] Parag Agrawal and Jennifer Widom. Confidence-aware joins in large uncertain databases. In *Stanford University Technical Report*, 2007.

[AY08] Charu Aggarwal and Philip Yu. On high dimensional indexing of uncertain data. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, pages 1460–1461, 2008.

[BAN08] Francesco Bonchi, Osman Abul, and Mirco Nanni. Never walk alone: Uncertainty for anonymity in moving object databases. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, pages 376–385, 2008.

[BDM$^+$05] Jihad Boulos, Nilesh Dalvi, Bhushan Mandhani, Shobhit Mathur, Chris Re, and Dan Suciu. MYSTIQ: A system for finding more answers by using probabilities. In *Proceedings of the SIGMOD/PODS Conference*, pages 891–893, 2005.

[BDRV05] Douglas Burdick, AnHai Doan, Raghu Ramakrishnan, and Shivakumar Vaithyanathan. OLAP over uncertain and imprecise data. In *VLDB 2005*, pages 39–50, 2005.

[Ben75] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[BGK$^+$07] Christian Böhm, Michael Gruber, Peter Kunath, Alexey Pryakhin, and Matthias Schubert. ProVeR: Probabilistic video retrieval using the Gauss-tree. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1521–1522, 2007.

[BGMP92] Daniel Barbará, Hector Garcia-Molina, and Daryl Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 4(5):487–502, 1992.

[BKS93] Thomas Brinkhoff, Hans-Peter Kriegel, and Bernhard Seeger. Efficient processing of spatial joins using r-trees. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 237–246, 1993.

[BKS01] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[BPS06a] Christian Böhm, Alexey Pryakhin, and Matthias Schubert. The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 9, 2006.

[BPS06b] Christian Böhm, Alexey Pryakhin, and Matthias Schubert. Probabilistic ranking queries on Gaussians. In *Proceedings of Statistical and Scientific DataBase Management (SSDBM)*, pages 169–178, 2006.

[BSHW06a] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, and Jennifer Widom. ULDBs: Databases with uncertainty and lineage. In *Proceed-*

ings of International Conference on Very Large Data Base (VLDB), pages 953–964, 2006.

[BSHW06b] Omar Benjelloun, Anish Das Sarma, Chris Hayworth, and Jennifer Widom. An introduction to ULDBs and the Trio system. *IEEE Data Engineering Bulletin*, 29(1):5–16, 2006.

[BSI08] George Bekales, Mohamed A. Soliman, and Ihab F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 326–339, 2008.

[CC07] Jinchuan Chen and Reynold Cheng. Efficient evaluation of imprecise location-dependent queries. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 586–595, 2007.

[CCCX09] Reynold Cheng, Lei Chen, Jinchuan Chen, and Xike Xie. Evaluating probability threshold k-nearest-neighbor queries over uncertain data. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 672–683, 2009.

[CCKN06] Michael Chau, Reynold Cheng, Ben Kao, and Jackey Ng. Uncertain data mining: An example in clustering location data. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 199–204, 2006.

[CCMC08] Reynold Cheng, Jinchuan Chen, Mohamed Mokbel, and Chi-Yin Chow. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 973–982, 2008.

[CCT96] Arbee L. P. Chen, Jui-Shang Chiu, and Frank S. C. Tseng. Evaluating aggregate operations over imprecise data. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 08(2):273–284, 1996.

[CG07] Graham Cormode and Minos Garofalakis. Sketching probabilistic data streams. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 281–292, 2007.

[CGGL03] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. Skyline with presorting. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 717–719, 2003.

[CJT$^+$06] Chee Yong Chan, H. V. Jagadish, Kian-Lee Tan, Anthony K. H. Tung, and Zhenjie Zhang. Finding k-dominant skylines in high dimensional space. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 503–514, 2006.

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

[CKP04] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. *IEEE Transactions on Data Engineering (TKDE)*, 16(9):1112–1127, 2004.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms 2nd Edition.* The MIT Press, 2001.

[CLY09] Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data and expected ranks. In *Proceedings of the In-*

ternational Conference on Data Engineering (ICDE), pages 305–316, 2009.

[CP03] Reynold Cheng and Sunil Prabhakar. Managing uncertainty in sensor databases. *SIGMOD Record*, 32(4):41–46, 2003.

[CR07] Dan Suciu Christopher Re, Nilesh N. Dalvi. Efficient top-k query evaluation on probabilistic data. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 886–895, 2007.

[CSP06] Reynold Cheng, Sarvjeet Singh, and Sunil Prabhakar. Efficient join processing over uncertain data. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM)*, pages 738–747, 2006.

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

[Din08] Zhiming Ding. UTR-tree: An index structure for the full uncertain trajectories of network-constrained moving objects. In *Proceedings of the International Conference on Mobile Data Management (MDM)*, pages 33–40, 2008.

[DM06] Amol Deshpande and Samuel Madden. Mauvedb: Supporting modelbased user views in database systems. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 73–84, 2006.

[DP98] Devdatt Dubhashi and Alessandro Panconesi. *Concentration* *of measure for the analysis of randomised algorithms, page 12.* http://citeseer.ist.psu.edu/old/ dubhashi98concentration.html, 1998.

[DS96] Debabrata Dey and Sumit Sarkar. A probabilistic relational model and algebra. *ACM Transactions on Database Systems (TODS)*, 21(2):339–369, 1996.

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

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

[DS07a] Nilesh Dalvi and Dan Suciu. The dichotomy of conjunctive queries on probabilistic structures. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 293–302, 2007.

[DS07b] Nilesh Dalvi and Dan Suciu. Management of probabilistic data: Foundations and challenges. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 1–12, 2007.

[DYM$^+$05] Xiangyuan Dai, Man Lung Yiu, Nikos Mamoulis, Yufei Tao, and Michail Vaitis. Probabilistic spatial queries on existentially uncertain data. In *Proceedings of the Symposium on Spatial and Temporal Databases (SSTD)*, pages 400–417, 2005.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of Knowledge Discovery and Data Mining Conference (KDD)*, pages 226–231, 1996.

[FB] Raphael A. Finkel and Jon Louis Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4(1).

[FHR08] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 656–665, 2008.

[FLN01] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. pages 102–113, 2001.

[FLN03] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66:614–656, 2003.

[FR07] Norbert Fuhr and Thomas Rolleke. A probabilistic NF2 relational algebra for imprecision in databases. In *Unpublished Manuscript*, 2007.

[FSR87] Christos Faloutsos, Timos K. Sellis, and Nick Roussopoulos. Analysis of object oriented spatial access methods. *ACM SIGMOD Record*, 16(3):426–439, 1987.

[Fuh90] Norbert Fuhr. A probabilistic framework for vague queries and imprecise information in databases. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 696–707, 1990.

[GJ90] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* New York, NY, USA, 1990.

[GJ00] Michael M. Guntzer and D. Jungnickel. Approximate minimization algorithms for the 0/1 knapsack and subset-sum problem. In *Operations Research Letters*, 2000.

[Gol] Oded Goldreich. Randomized methods in computation. `http://www.wisdom.weizmann.ac.il/~oded/rnd.html`.

[Gol01] Oded Goldreich. *Randomized Methods in Computation, Lecture 2*. http://www.wisdom.weizmann.ac.il/~oded/rnd.html, 2001.

[GUP05] Jose Galindo, Angelica Urrutia, and Mario Piattini. Fuzzy databases: Modeling, design, and implementation., 2005.

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

[GZM09] Tingjian Ge, Stan Zdonik, and Samuel Madden. Top-k queries on uncertain data: On score distribution and typical answers. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 375–388, 2009.

[HGS03a] Edward Hung, Lise Getoor, and V. S. Subrahmanian. Probabilistic interval XML. In *Proceedings of the International Conference on Database Theory (ICDT)*, pages 358–374, 2003.

[HGS03b] Edward Hung, Lise Getoor, and V. S. Subrahmanian. PXML: A probabilistic semistructured data model and algebra. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 467, 2003.

[HHLM07] Ramaswamy Hariharan, Bijit Hore, Chen Li, and Sharad Mehrotra. Processing spatial-keyword (sk) queries in geographic information retrieval (gir) systems. In *Proceedings of Statistical and Scientific DataBase Management (SSDBM)*, page 16, 2007.

[HJLO06] Zhiyong Huang, Christian S. Jensen, Hua Li, and Beng Chin Ooi. Skyline queries against mobile lightweight devices in MANETs. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, page 66, 2006.

[HJR97] Yun-Wu Huang, Ning Jing, and Elke A. Rundensteiner. Spatial joins using r-trees: Breadth-first traversal with global optimizations. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 396–405, 1997.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques.* 2001.

[Hoe63] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *The Journal of the Acoustical Society of America-Online (JASA)*, 1963.

[HPZL08a] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *Proceedings of the Twenty-fourth International Conference on Data Engineering*, pages 1403–1405, 2008.

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

[HS89] John Hershberger and Subhash Suri. Finding tailored partitions. In *Proceedings of the Symposium on Computational Geometry*, pages 255–265, 1989.

[HS95] Gísli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *SSD*, pages 83–95, 1995.

[HS99] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems (TODS)*, (2):265–318, 1999.

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

[JKV07] T. S. Jayram, Satyen Kale, and Erik Vee. Efficient aggregation algorithms for probabilistic data. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 346–355, 2007.

[JYC$^+$08] Cheqing Jin, Ke Yi, Lei Chen, Jeffrey Xu Yu, and Xuemin Lin. Sliding-window top-k queries on uncertain streams. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 301–312, 2008.

[KHKH07] Dong-Oh Kim, Dong-Suk Hong, Hong-Koo Kang, and Ki-Joon Han. Ur-tree: An efficient index for uncertain data in ubiquitous sensor networks. In *Advances in Grid and Pervasive Computing*, pages 603–613, 2007.

[KKPR06] Hans-Peter Kriegel, Peter Kunath, Martin Pfeifle, and Matthias Renz. Probabilistic similarity join on uncertain data. In *Proceedings of Database Systems for Advanced Applications (DASFAA)*, pages 295–309, 2006.

[KKR07] Hans-Peter Kriegel, Peter Kunath, and Matthias Renz. Probabilistic nearest-neighbor query on uncertain objects. In *Proceedings of Database Systems for Advanced Applications (DASFAA)*, pages 337–348, 2007.

[KP05] Hans-Peter Kriegel and Martin Pfeifle. Density-based clustering of uncertain data. In *Proceedings of Knowledge Discovery and Data Mining Conference (KDD)*, pages 672–677, 2005.

[KS95] Ron Kohavi and Dan Sommerfield. Feature subset selection using the wrapper model: Overfitting and dynamic search space topology. In *Proceedings of Knowledge Discovery and Data Mining Conference (KDD)*, pages 192–197, 1995.

[KS07] Benny Kimelfeld and Yehoshua Sagiv. Matching twigs in probabilistic XML. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 27–38, 2007.

[KW86] Marlvin H. Kalos and Paula A. Whitlock. *Monte Carlo Methods.* Wiley Interscience, 1986.

[LBR$^+$07] Rui Li, Bir Bhanu, Chinya Ravishankar, Michael Kurth, and Jinfeng Ni. Uncertain spatial data handling: Modeling, indexing and query. *Computational Geoscience*, 33(1):42–61, 2007.

[LC08] Xiang Lian and Lei Chen. Monochromatic and bichromatic reverse skyline search over uncertain databases. In *The Proceedings of ACM SIGMOD/PODS Conference*, pages 213–226, 2008.

[LCLC04] Kam-Yiu Lam, Reynold Cheng, Biyu Liang, and Jo Chau. Sensor node selection for execution of continuous probabilistic queries in wireless sensor networks. In *Proceedings of the ACM International Workshop on Video Surveillance and Sensor Networks*, pages 63–71, 2004.

[Lee92] Suk Kyoon Lee. Imprecise and uncertain information in databases: an evidential approach. In *Proceedings of the Eighth Intertional Conference on Data Engineering (ICDE)*, pages 614–621, 1992.

[LLRS97] Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and Siva Subrahmanian. ProbView: a flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 22(3):419–469, 1997.

[LSD09] Jian Li, Barna Saha, and Amol Deshpande. A unified approach to ranking in probabilistic databases. pages 502–513, 2009.

[LSS96] Ee-Peng Lim, Jaideep Srivastava, and Shashi Shekhar. An evidential reasoning approach to attribute value conflict resolution in database integration. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):707–723, 1996.

[LYWL05] Xuemin Lin, Yidong Yuan, Wei Wang, and Hongjun Lu. Stabbing the sky: Efficient skyline computation over sliding windows. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, pages 502–513, 2005.

[MKM08] Yiming Ma, Dmitri V. Kalashnikov, and Sharad Mehrotra. Toward managing uncertain spatial information for situational awareness applications. *IEEE Transactions on Knowledge and Data Engineering*, 20(10):1408–1423, 2008.

[Mot88] Amihai Motro. Vague: a user interface to relational databases that permits vague queries. *ACM Transactions on Information Systems*, 6(3):187–214, 1988.

[MSCP03] Chris Mayfield, Sarvjeet Singh, Reynold Cheng, and Sunil Prabhakar. ORION: A database system for managing uncertain data., 2003.

[MSS01] Sally McClean, Bryan Scotney, and Mary Shapcott. Aggregation of imprecise and uncertain information in databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 13(6):902–912, 2001.

[MTdK$^+$07] Michi Mutsuzaki, Martin Theobald, Ander de Keijzer, Jennifer Widom, Parag Agrawal, Omar Benjelloun, Anish Das Sarma, Raghotham Murthy, and Tomoe Sugihara. TrioOne: Layering uncertainty and lineage on a conventional dbms. In *Proceedings of Conference on Innovative Database Systems Research (CIDR)*, pages 269–274, 2007.

[MW07] Raghotham Murthy and Jennifer Widom. Making aggregation work in uncertain and probabilistic databases. In *Workshop on Management of Uncertain Data 2007*, 2007.

[NJ02] Andrew Nierman and H. V. Jagadish. ProTDB: Probabilistic data in XML. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 646–657, 2002.

[NKC$^+$06] Wang Kay Ngai, Ben Kao, Chun Kit Chui, Reynold Cheng, Michael Chau, and Kevin Y. Yip. Efficient clustering of uncertain data. In *Proceedings of the International Conference on Data Mining (ICDM)*, pages 436–445, 2006.

[Pea88] J. Pearl. Probabilistic reasoning in intelligent systems. In *Morgan Kaufmann Publishers*, 1988.

[PHTL] Jian Pei, Ming Hua, Yufei Tao, and Xuemin Lin. Query answering technique on uncertain and probabilistic data. In *The Proceedings of SIGMOD/PODS Conference*.

[PJET05] Jian Pei, Wen Jin, Martin Ester, and Yufei Tao. Catching the best views of skyline: A semantic approach based on decisive subspaces. In *Proceedings of International Conference on Very Large Data Base (VLDB)*, pages 253–264, 2005.

[PJLY07] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. Probabilistic skyline on uncertain data. In *Proceedings of the Thirty-third Very Large Data Base (VLDB) Conference*, pages 15–26, 2007.

[PKZT01] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *Proceedings of the Symposium on Spatial and Temporal Databases (SSTD)*, pages 443–459, 2001.

[PMT99] Dimitris Papadias, Nikos Mamoulis, and Yannis Theodoridis. Processing and optimization of multiway spatial joins using R-trees. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 44–55, 1999.

[PTFS03] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. An optimal and progressive algorithm for skyline queries. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 467–478, 2003.

[PTGS03] D. Papadias, Y. Tao, F. Greg, and B. Seeger. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)*, 30(1):41–82, 2003.

[RB92] Elke A. Rundensteiner and Lubomir Bic. Evaluating aggregates in possibilistic relational databases. *Data and Knowledge Engineering (DKE)*, 7(3):239–267, 1992.

[RKF95] Nick Roussopoulos, Stephen Kelley, and Frederick. Nearest neighbor queries. In *Proceedings of the ACM SIGMOD Conference*, pages 71–79, 1995.

[RSG05] Robert B. Ross, V. S. Subrahmanian, and John Grant. Aggregate operators in probabilistic databases. *Journal of ACM*, 52(1):54–101, 2005.

[SA07] Pierre Senellart and Serge Abiteboul. On the complexity of managing probabilistic XML data. In *Proceedings of the ACM SIGMOD/PODS Conference*, pages 283–292, 2007.

[Sad05] Yukio Sadahiro. Buffer operations on spatial data with limited accuracy. *Transactions in GIS*, 9(3):323–344, 2005.

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

[SD07] Prithviraj Sen and Amol Deshpande. Representing and querying correlated tuples in probabilistic databases. In *Proceedings of International Conference on Data Engineering (ICDE)*, pages 596–605, 2007.

[SFC07] Mohamed A. Soliman, Ihab F.Ilyas, and Kevin Chen-Chuan Chang. URank: Formulation and efficient evaluation of top-k queries in uncertain databases. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 1082–1084, 2007.

[SIC07] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *Proceedings of International Conference on Data Engineering*, pages 896–905, 2007.

[SM03] Bryan W. Scotney and Sally I. McClean. Database aggregation of imprecise and uncertain evidence. *Inf. Sci.*, 155(3):245–263, 2003.

[SMM$^+$08] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne Hambrusch, and Rahul Shah. Orion 2.0: Native support for uncertain data. In *Proceedings of ACM SIGMOD/PODS Conference*, pages 1239–1242, 2008.

[SMP$^+$07] Sarvjeet Singh, Chris Mayfield, Sunil Prabhakar, Rahul Shah, and Susanne E. Hambrusch. Indexing uncertain categorical data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 616–625, 2007.

[SS06] Mehdi Sharifzadeh and Cyrus Shahabi. The spatial skyline queries. In *Proceedings of International Conference on Very Large Data Base (VLDB)*, pages 751–762, 2006.

[STW08] Anish Das Sarma, Martin Theobald, and Jennifer Widom. Exploiting lineage for confidence computation in uncertain and probabilistic databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1023–1032, 2008.

[SUW08] Anish Das Sarma, Jeffrey Ullman, and Jennifer Widom. Functional dependencies for uncertain relations. In *ICDE 2008*, 2008.

[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. pages 922–933, 2005.

[TEO01] Kian-Lee Tan, P. Eng, and Beng Chin Ooi. Efficient progressive skyline computation. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 301–310, 2001.

[TP06] Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. In *IEEE Transactions on Knowledge and Data Engineering*, volume 18, pages 377–391, 2006.

[TS96] Yannis Theodoridis and Timos K. Sellis. A model for the prediction of r-tree performance. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 161–171, 1996.

[TXC07] Yufei Tao, Xiaokui Xiao, and Reynold Cheng. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems*, 32(3):15, 2007.

[TXP06] Yufei Tao, Xiaokui Xiao, and Jian Pei. SUBSKY: Efficient computation of skylines in subspaces. In *Proceedings of the Twenty-fourth International Conference on Data Engineering (ICDE)*, page 65, 2006.

[Wid05] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of Conference on Innovative Database Systems Research (CIDR)*, pages 262–276, 2005.

[YLKS08] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. Efficient processing of top-k queries in uncertain databases with x-relations. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 20(12):1669–1682, 2008.

[YLL$^+$05] Yidong Yuan, Xuemin Lin, Qing Liu, Wei Wang, Jeffrey Xu Yu, and Qing Zhang. Efficient computation of the skyline cube. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 241–252, 2005.

[YM07] Man Lung Yiu and Nikos Mamoulis. Efficient processing of top-k dominating queries on multi-dimensional data. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 483–494, 2007.

[YMT06] Man Lung Yiu, Nikos Mamoulis, and Yufei Tao. Efficient quantile retrieval on multi-dimensional data. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 167–185, 2006.

[ZBG07] Daniel Zinn, Jim Bosch, and Michael Gertz. Modeling and querying vague spatial objects using shapelets. In *Proceedings of the International Conference on Very Large Data Base (VLDB)*, pages 567–578, 2007.

[ZLZ$^+$09] Wenjie Zhang, Xuemin Lin, Ying Zhang, Jian Pei, and Wei Wang. Threshold-based probabilistic top-k dominating queries. In *to appear in VLDB Journal*, 2009.

[ZRL96] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: An efficient data clustering method for very large databases. In *Proceedings of the ACM SIGMOD Conference*, pages 103–114, 1996.

Appendix A

Related Publications and Works

1. Ying Zhang, Wenjie Zhang, Xuemin Lin, Jian Pei and Bin Jiang, "Ranking Uncertain Sky: the Probabilistic Top-k Skyline Operator", to appear in Information Systems Journal. (Accepted 2010)

2. Ying Zhang, Xuemin Lin, Wenjie Zhang, Jianmin Wang and Qianlu Lin, "Effectively Indexing the Uncertain Space", to appear in IEEE Transactions on Knowledge and Data Engineering (TKDE). (Accepted 2009).

3. Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, Ying Zhang, Wei Wang, "Quantile-Based KNN Over Multi-Valued Objects", in the Proceedings of International Conference on Data Engineering (ICDE), 16-27, 2010.

4. Wenjie Zhang, Xuemin Lin, Ying Zhang, Jian Pei and Wei Wang, "Threshold-based Probabilistic Top-k Dominating Queries", VLDB Journal 19(2), 283-305, 2010.

5. Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, Jeffrey Xu Yu, "Probabilistic Skyline Operator over Sliding Windows", in the Proceedings of International Conference on Data Engineering (ICDE), 1060-1071, 2009.

6. Wenjie Zhang, Xuemin Lin, Jian Pei, Ying Zhang, “Managing Uncertain Data: Probabilistic Approaches”, in the Proceedings of the International Conference on Web-Age Information Management (WAIM), 405-412, 2008. (**invited paper**)

Appendix B

Academic Achievements

Journal Articles

1. Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang and Wei Wang, "Continuous Monitoring of Distance Based Range Queries", submitted to IEEE Transactions on Data Engineering (TKDE). (Accepted 2010)

2. Ying Zhang, Wenjie Zhang, Xuemin Lin, Jian Pei and Bin Jiang, "Ranking Uncertain Sky: the Probabilistic Top-k Skyline Operator", to appear in Information Systems Journal. (Accepted 2010)

3. Ying Zhang, Xuemin Lin, Wenjie Zhang, Jianmin Wang and Qianlu Lin, "Effectively Indexing the Uncertain Space", to appear in IEEE Transactions on Knowledge and Data Engineering (TKDE). (Accepted 2009)

4. Wenjie Zhang, Xuemin Lin, Ying Zhang, Jian Pei and Wei Wang, "Threshold-based Probabilistic Top-k Dominating Queries", VLDB Journal 19(2), 283-305, 2010.

5. Muhammad Aamir Cheema, Xuemin Lin, Wei Wang, Wenjie Zhang, Jian Pei, "Probabilistic Reverse Nearest Neighbor Queries on Uncertain Data", IEEE

Transactions on Knowledge and Data Engineering (TKDE) 22(4), 550-564, 2010.

Conference Articles

1. Gaoping Zhu, Xuemin Lin, Wenjie Zhang, Wei Wang, Haichuan Shang, "PrefIndex: An Efficient Supergraph Containment Search Technique", in the Proceedings of Scientific and Statistical Database Management Conference (SSDBM), 360-378, 2010.

2. Wenjie Zhang, Ying Zhang, Muhammad Aamir Cheema, Xuemin Lin, "Counting Distinct Objects over Sliding Windows", in the Proceedings of Australasian Database Conference (ADC) 2010. (**Best Paper Award**)

3. Wenjie Zhang, Xuemin Lin, Muhammad Aamir Cheema, Ying Zhang, Wei Wang, "Quantile-Based KNN Over Multi-Valued Objects", in the Proceedings of International Conference on Data Engineering (ICDE), 16-27, 2010.

4. Ying Zhang, Xuemin Lin, Gaoping Zhu, Wenjie Zhang, Qianlu Lin, "Efficient Rank Based KNN Processing over Uncertain Data", in the Proceedings of International Conference on Data Engineering (ICDE), 28-39, 2010.

5. Muhammad Aamir Cheema, Ljiljana Brankovic, Xuemin Lin, Wenjie Zhang, Wei Wang, "Multi-Guarded Safe Zone: An Efficient Technique to Monitor Moving Circular Range Queries", in the Proceedings of International Conference on Data Engineering (ICDE), 189-200, 2010.

6. Muhammad Aamir Cheema, Xuemin Lin, Ying Zhang, Wei Wang, Wenjie Zhang, "Lazy Updates: An Efficient Technique to Continuously Monitoring Reverse kNN", in the Proceedings of International Conference on Very Large Data Bases (VLDB), PVLDB 2(1), 1138-1149, 2009.

7. Shuxiang Yang, Wenjie Zhang, Ying Zhang, Xuemin Lin, "Probabilistic Threshold Range Aggregate Query Processing over Uncertain Data", in the Proceedings of the Joint International Conferences on Asia-Pacific Web Conference (APWeb) and Web-Age Information Management (WAIM) APWeb-WAIM, 51-62, 2009. (**Best Paper Award**)

8. Wenjie Zhang, Xuemin Lin, Ying Zhang, Wei Wang, Jeffrey Xu Yu, "Probabilistic Skyline Operator over Sliding Windows", in the Proceedings of International Conference on Data Engineering (ICDE), 1060-1071, 2009.

9. Ying Zhang, Xuemin Lin, Yufei Tao, Wenjie Zhang, "Uncertain Location based Range Aggregates in a Multi-Dimensional Space", in the Proceedings of International Conference on Data Engineering (ICDE), 1247-1250, 2009. (short paper)

10. Wenjie Zhang, Xuemin Lin, Jian Pei, Ying Zhang, "Managing Uncertain Data: Probabilistic Approaches", in the Proceedings of the International Conference on Web-Age Information Management (WAIM), 405-412, 2008. (**invited paper**)

11. Ming Hua, Jian Pei, Wenjie Zhang, Xuemin Lin, "Ranking Queries on Uncertain Data: A Probabilistic Threshold Approach", in the Proceedings of ACM SIGMOD Conference, 673-686, 2008.

12. Ming Hua, Jian Pei, Wenjie Zhang, Xuemin Lin, "Efficiently Answering Probabilistic Threshold Top-k Queries on Uncertain Data", in the Proceedings of International Conference on Data Engineering (ICDE), 1403-1405, 2008. (short paper)