Query Processing over Probabilistic Data with Gaussian Distributions

Tingting DONG

Abstract

The field of *uncertain data management* has received extensive attention of researchers due to the increasing demand for managing *uncertain data* in a large variety of real-world applications such as sensor networks, location-based services, monitoring and surveillance. Uncertainty can occur for different reasons, including measurement errors and noises in sensors, privacy-preserving transformations of sensitive records and the limited confidence in the output of predictive models.

Managing uncertainty involves modeling, representing, querying, and indexing uncertain data. Answering queries over uncertain databases poses more challenges than that over traditional databases because managing uncertainty usually means costly probability computations. Hence, it is crucial to develop efficient solutions when managing uncertain data. In this thesis, we model uncertainty probabilistically and represent each uncertain object in the database using a *Gaussian distribution*, which is a typical probability distribution widely used in statistics, pattern recognition, and machine learning. We consider the following three types of queries or searches over probabilistic data with Gaussian distributions.

First, we study the probabilistic range query, which is an important query in the field of uncertain data management. A probabilistic range query returns objects in the database that exist within a specified range from the query object with probabilities no less than a given probability threshold. The query object can be either a certain point or an uncertain object represented by a Gaussian distribution.

We propose several effective filtering techniques by analyzing the properties of Gaussian distribution. The proposed filtering techniques can significantly reduce the number of objects that need to be verified by expensive probability computation. In this way, we can avoid unnecessary computations and hence save cost. To support efficient query processing, we further propose a novel indexing method to enable filtering unpromising objects by groups rather than individually. We develop the indexing method by extending the existing R-tree and improve it based on our analysis of Gaussian distribution. This indexing method can effectively organize objects and greatly enhance the performance of query processing. Extensive experiments on real datasets demonstrate the efficiency and effectiveness of our proposed approach.

Second, we investigate the *nearest neighbor search*. As one of the commonest queries over location information, the distance-based nearest neighbor search, which finds closest

objects to a given query point, has extensive applications in many areas. There have been considerable efforts to extend nearest neighbor search over traditional location information to uncertain location information. An example is the *expected distance*, which defines the distance over uncertain location information. Following this trend, we represent uncertain locations using Gaussian distributions and assume that the closeness between each Gaussian object and the query point is measured by their expected distance. Under this setting, we consider the problem of *k-expected nearest neighbor search* over Gaussian objects. The result objects are ones that have the top-*k* smallest expected distances to the query point.

We analyze properties of expected distance on Gaussian distribution mathematically and derive the lower bound and upper bound of the distance. Based on our analysis, we propose three novel approaches to efficiently solve this problem. The proposed approaches can prune unpromising objects whose lower bound distances are larger than upper bound or expected distances of candidate objects without computing their actual expected distances. We only compute exact expected distances for candidate objects and finally return the top-*k* smallest ones. To further improve the performance, we utilize R-tree to index objects and their lower bound distances and upper bound distances. The proposed approaches can effectively reduce the number of exact distance computation which is rather expensive. The efficiency and effectiveness of our approaches are demonstrated through extensive experiments.

Finally, we explore the problem of similarity search, which is a crucial task in many real-world applications such as multimedia databases, data mining, and bioinformatics. In this work, we investigate similarity search on uncertain data represented by Gaussian distributions. The query object is also represented by a Gaussian distribution. By employing *Kullback-Leibler divergence* (KL-divergence) to measure the similarity between two Gaussian distributions, our goal is to search a database for the top-*k* Gaussian distributions similar to a given query Gaussian distribution. Especially, we consider non-correlated Gaussian distributions, where there are no correlations between dimensions and their covariance matrices are diagonal.

To support query processing, we propose two types of novel approaches utilizing the notions of *rank aggregation* and *skyline queries*. The first type presorts all objects in the database on their attributes and computes result objects by merging candidates from each presorted list. The second one transforms the problem to the computation of *dynamic skyline queries*. We extend and modify the branch-and-bound skyline (BBS) algorithm, which is proposed to answer skyline queries, and develop a novel algorithm to solve this problem. We demonstrate the efficiency and effectiveness of our approaches through a comprehensive experimental performance study.

In general, we provide a comprehensive view of managing probabilistic data with Gaussian distributions. We believe that our contributions are multi-dimensional and will be extended over time. First of all, we think our mathematical analyses of probabilistic range query, nearest neighbor search, and similarity search over Gaussian objects will benefit not only the existing related applications, but also potential studies of other applications over data represented by Gaussian distributions. Furthermore, our proposed non-trivial algorithms and indexing structures for efficient query processing, will not only enrich user experience by speed in the real world, but also provide valuable insights and references for developing solutions to other problems. Last but not least, we have conducted extensive experimental evaluations on the performance of our proposed solutions.

Acknowledgements

First of all, I would like to express my sincere gratitude to my supervisor Prof. Yoshiharu Ishikawa for the continuous support of my Ph.D. studies. His valuable academic guidance and constant encouragement helped me all the way to the completion of this thesis. Meanwhile, his patience, honesty, diligence, enthusiasm and immense knowledge not only impressed me greatly, but also enlightened my future career.

I would also like to thank the rest of my thesis committee: Prof. Kenji Mase and Prof. Katsuhiko Toyama, for their careful reading, insightful comments, and fruitful discussions.

My sincere thanks also go to Prof. Chuan Xiao and Dr. Xi Guo, co-authors of papers I published during my Ph.D. studies, for their constructive advice and thoughtful discussions.

I would like to take this opportunity to express my gratitude to Prof. Hiroyuki Kitagawa, Prof. Lei Chen, Prof. Heng Tao Shen, and Prof. Jeffrey Xu Yu, for their unceasing encouragement and various forms of support.

I am also very grateful to the other current and past members of our laboratory at Nagoya University. Their kindness and friendship brought my unforgettable experience in this laboratory and enriched my life.

Last but not least, I would like to thank my parents, for raising me and supporting me spiritually throughout my life. I would like to thank my friends who make my life merry and full of happiness as well.

Contents

List of Figures

List of Tables

Abbreviations

BB	**B**ounding **B**ox
BBS	**B**ranch-and-**B**ound **S**kyline
ESQED	**E**xpected **SQ**uared **E**uclidean **D**istance
LB	**L**ower **B**ound
MBB	**M**inimum **B**ounding **B**ox
MBR	**M**inimum **B**ounding **R**ectangle
PRQ	**P**robabilistic **R**ange **Q**uery
PRQ-P	**P**robabilistic **R**ange **Q**uery with a **P**oint query object
PRQ-G	**P**robabilistic **R**ange **Q**uery with a **G**aussian query object
QR	**Q**uery **R**egion
SQED	**SQ**uared **E**uclidean **D**istance
UB	**U**pper **B**ound

Chapter 1

Introduction

1.1 Research Background

In recent years, the rapid advancement in data acquiring devices has led to the increasing availability of massive data in a wide variety of fields such as information retrieval, data integration, mobile robotics, sensor networks, location-based services, monitoring and surveillance. One common characteristic of the data acquired in these fields is their *uncertain* nature. The reasons for *uncertainty* in data are diverse and ubiquitous shown as follows [1, 2].

1. The uncertainty may be a result of the inherent limitations of the underlying equipment. For example, in sensor and RFID data, uncertainty is due to measurement errors and noises. The output of sensor networks is uncertain because of the noise in sensor inputs and errors in wireless transmission [3].

2. In some applications, such as privacy-preserving data mining [4], it is a requirement that the sensitive data be less precise. For instance, the data is perturbed purposely to preserve the sensitivity of attribute values so that the data can be published [5].

3. In many cases, uncertainty results from the effect of statistical methods such as forecasting and imputation or domain-specific properties and techniques. In the area of probabilistic robotics [6], the behavior or location of a robot is usually

estimated using the Markov or Bayes theory which forms a source of uncertainty due to the limited confidence in the output of predictive models. In information extraction, uncertainty comes from the imperfection of extraction tools and the inherent ambiguity in unstructured text [7].

In some cases, it is possible to eliminate the uncertainties completely. However, this is usually very costly, like manual removal of ambiguous matches in data cleaning. In many cases, complete removal is not even possible. *Uncertain data* has no place in traditional, precise database management systems like inventory and employee. This has created a need for *uncertain data management*.

Following this trend, the database and data mining community has investigated extensively the problem of modeling and querying uncertain data [8, 9] in recent decades. Hundreds of research work has been done on this topic [1, 10] and several probabilistic database management systems have been proposed [11–13].

The general approach to manage uncertain data is with a probabilistic model [14], which represents uncertainties using probabilities. According to different granularities, Wang *et al.* [10] classify uncertainty by three levels: table-based, tuple-based, and attribute-based, with the latter two ones have been widely studied in the literature. The tuple-level uncertainty describes the probability that a tuple exists in the database. On the other hand, the attribute-level uncertainty indicates the probability that a specific value is randomly selected to represent an individual attribute of a tuple from many alternative values. For example, in a database of patients, the disease of a patient can be a cold with a probability of 0.7 and a flu with a probability of 0.3. In this case, the attribute disease is represented by a discrete probability distribution $\{0.7, 0.3\}$. If the attribute has a continuous domain like the location of a mobile robot, a continuous probability density function such as *Gaussian distribution* is often utilized to represent this kind of uncertainty.

In this thesis, we focus on the continuous attribute uncertainty and the case where uncertainty is represented by Gaussian distribution. In other words, we assume that uncertain objects stored in the database are represented by Gaussian distributions. Under this setting, we consider query processing over probabilistic data with Gaussian distributions.

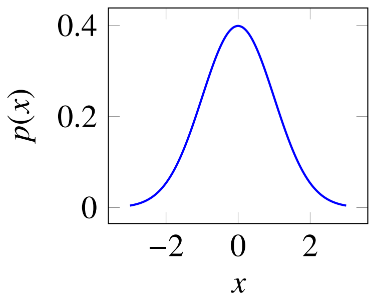


FIGURE 1.1: One-dimensional Gaussian distribution

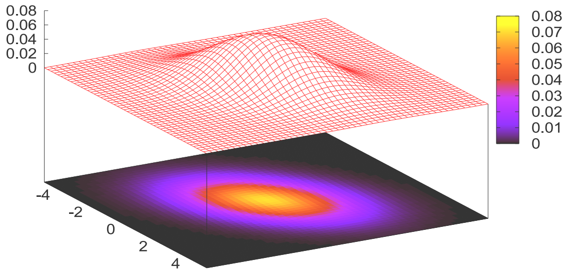


FIGURE 1.2: Two-dimensional Gaussian distribution

We focus our effort on Gaussian distribution because it is one of the most typical probability distributions, and is widely used in statistics, pattern recognition, and machine learning [15, 16]. Especially, in the area of spatio-temporal databases, it is frequently used to represent uncertain location information [14, 17–19].

In Fig. 1.1 and Fig. 1.2, we show examples of one-dimensional and two-dimensional Gaussian distribution, respectively. Generally speaking, it describes the probability that a random point distributes in the space. For example, from Fig. 1.1 in the one-dimensional space, the probability of a point being around the center 0 is about 0.4. The probability decreases as the point spreads from the center. This is the similar in the two-dimensional space in Fig. 1.2. We project the probability surface to a plane and show the decreasing trend with gradient colors. When using Gaussian distribution to represent uncertain location information, it means that the object has the highest probability to be located in the reported location (i.e., the center) and the farther it is, the smaller the probability to be located there becomes.

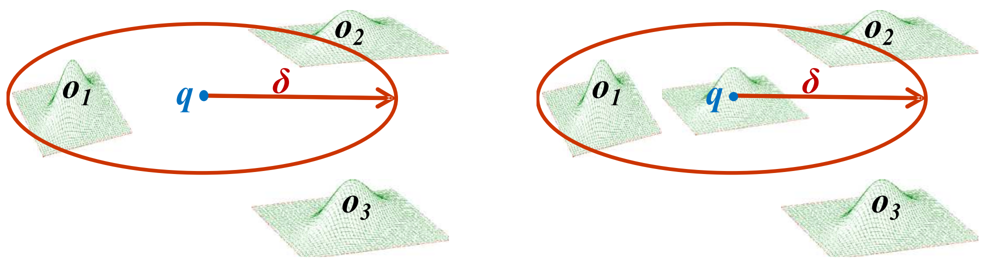
In the first work, we consider *range query*, one of the most important queries over spatio-temporal databases. A range query retrieves all the objects that are within the given search range. A range query over uncertain objects, called a *probabilistic range query* [19, 20], searches for objects within the given search range with probabilities no less than a specified probability threshold. The query object can be either a certain point or an uncertain object represented by a Gaussian distribution. In location-based services, when representing the locations of landmarks by probability distributions like Gaussian distributions, this kind of query can be used to find surrounding landmarks for a self-navigated mobile robot when building the environment map.

Our second work studies *nearest neighbor query* or *nearest neighbor search*, which is also a common type of query in spatio-temporal databases. This query returns the nearest objects to a query point. When extending traditional certain objects to uncertain objects, there are two variations for the new query. The first one, called *probabilistic nearest neighbor query* [21], utilizes a probability threshold to define qualifying objects in a similar way as the range query. The second one, called *expected nearest neighbor search*, identifies objects based on their *expected distances* [22, 23] to the query point. We focus on the second one and consider *k*-expected nearest neighbor search over uncertain objects represented by Gaussian distributions. An application example is to find potential customers nearby for shops or restaurants.

The third work explores *similarity search* over Gaussian distributions. Given a database of Gaussian distributions and a query Gaussian distribution, we search the database for top similar data Gaussian distributions. We utilize the *Kullback-Leibler divergence* (KL-divergence) to measure the similarity between two Gaussian distributions. As in [24], we can represent feature vectors generated in pattern recognition and machine learning using Gaussian distributions and conduct similarity search over them to make interesting and useful findings.

1.2 Research Objectives and Contributions

We have conducted three work on probabilistic data with Gaussian distributions. Our first and second work are motivated by the uncertainty in location information in the area of spatial databases. We represent uncertain locations of objects in the database using Gaussian distributions. In the first work, we assume two kinds of query objects: a certain point and an uncertain location. If the query object is uncertain, we also represent it using a Gaussian distribution. We call them point and Gaussian query object, respectively. Probabilistic range queries by the two kinds of query objects are different and should be handled separately. We show their examples in Fig. 1.3. Here, we define the search



(a) Point query object (b) Gaussian query object

FIGURE 1.3: Examples of probabilistic range queries

region as the circular area with the query object q as the center and radius δ. Given a query object q, a range δ, and a probability threshold θ, we search for the objects located within δ from q with probabilities at least θ.

Although there have been solutions to probabilistic range queries that can handle probabilistic data with Gaussian distributions [25, 26], they are based on specific assumptions and cannot be used to solve the problem in this work. The naïve approach, which sequentially verifies all the objects in the database by computing their exact probabilities, is prohibitively expensive for most real-world applications. This challenge motivates us to develop efficient algorithms to support query processing. We propose a set of filtering techniques to avoid unnecessary computations and a novel indexing method to accelerate query processing. The proposed filtering techniques are based on properties of Gaussian distribution and very effective in reducing computation cost. Due to these techniques, only a small part of objects need to be checked. We develop the indexing method by extending the existing R-tree and improve it based on our analysis of Gaussian distribution. The indexing method organizes all the objects in a structural way so that we can process them by groups efficiently rather than individually.

In the second work, we assume only the point query object. The expected nearest neighbor search returns objects that are close to the query point based on their expected distances. Since most users are interested in the top result objects, we consider k-expected nearest neighbor search and returns the top-k results to users. There is no previous work on k-expected nearest neighbor search over objects represented by Gaussian distributions. Moreover, the naïve approach of performing sequential scan is also computationally expensive for the problem in this work. To explore an efficient solution, we analyze properties of expected distance on Gaussian distribution mathematically and derive the lower bound and upper bound of this distance. We develop effective filtering techniques using the lower bound and upper bound. The proposed filtering techniques can prune unpromising objects whose lower bound distances are larger than upper bound or expected distances of candidate objects without computing their actual expected distances. To further improve the performance, we utilize R-tree to index all the objects and their lower bound distances and upper bound distances. We propose three novel algorithms to support efficient query processing. The proposed approaches can effectively reduce the number of expensive distance computation.

The third work takes a different view and focuses on the property of Gaussian distribution of being a probability distribution. We use Gaussian distribution as the query object and search for top-*k* similar Gaussian distributions in the database. The similarity between two Gaussian distributions here is measured by *Kullback-Leibler divergence* (KL-divergence) [27], which is a representative measure for quantifying the similarity between two probability distributions. Since KL-divergence is a non-metric measure which violates the properties of a standard distance function in metric spaces, existing solutions for metric spaces cannot be employed to solve this problem. Although there have been several studies on searching in non-metric spaces [28–31], their problem settings are different from ours. The naïve solution of sequential scan is again not suitable for this problem. To this end, We propose two types of approaches utilizing the notions of *rank aggregation* [32] and *skyline queries* [33]. The first type presorts all objects in the database on their attributes and computes result objects by merging candidates from each presorted list. The second one transforms the problem to the computation of *dynamic skyline queries*. We extend and modify the branch-and-bound skyline (BBS) algorithm, which is proposed to answer skyline queries, and develop a novel algorithm to solve this problem. We propose novel algorithms to support efficient query processing by modifying and extending existing algorithms.

Our contributions are summarized as follows:

- We formalize two types of probabilistic range queries with respect to the point and Gaussian query object.

- For the two types of probabilistic range queries, we propose effective filtering techniques and a novel indexing method to support efficient query processing.
- We formally define the problem of k-expected nearest neighbor search over objects represented by Gaussian distributions.
- We analyze mathematically properties of the expected distance on Gaussian distribution and derive the lower bound and upper bound of this distance.
- We propose three novel approaches to improve the efficiency of k-expected nearest neighbor search.
- We formalize the problem of top-k similarity search based on KL-divergence over Gaussian distributions, and analyze mathematically the properties of KL-divergence between two Gaussian distributions.
- We propose two types of approaches to improve the efficiency of query processing using the notion of rank aggregation and skyline queries.
- We demonstrate the efficiency and effectiveness of our approaches through comprehensive experimental performance studies.

1.3 Related Work

The field of uncertain data management has received considerable attention from many researchers since its growing up in recent decades. Managing uncertainty poses a number of unique challenges on several fronts. The two broad issues are those of modeling the uncertain data, and then leveraging it to work with various applications [1]. In [34, 35], several issues and working models for uncertain data have been discussed. The other issue is that of adapting data management and mining applications for uncertain data. There have been a number of excellent surveys and books discussing uncertain data management [1, 9, 10, 34, 36, 37]. Li *et al.* [38] provides a cross-disciplinary view of uncertainty processing activities by different communities.

Several models have been proposed to incorporate uncertain objects in databases [14], including the fuzzy model [39], the evidence-oriented model [40], and the probabilistic model. Moreover, probabilistic graphical models [41] are proposed to represent complex dependencies among uncertain data. These models are mainly different in the semantics and complexities of the data in the underlying applications [35]. In the area of database, the general approach to manage uncertain data is the use of probabilistic model, which represents uncertainties using probabilities. According to different granularities of uncertainty, the probabilistic model can be further classified into three categories: table-based, tuple-based, and attribute-based [10, 14], with the latter two ones have been widely studied in the literature.

Specifically, a *table-based* approach concerns the “coverage” of a table, i.e., the percentage of tuples present in a table [42]. On the other hand, a *tuple-based* solution associates each tuple with a probability indicating the likelihood that the tuple exists in the table [43]. An *attribute-based* method represents the attribute of a tuple using a probability distribution [3, 19, 44, 45]. The probability distribution can be either a discrete one, which describes a set of possible values with occurring probabilities [3, 44], or a continuous one such as a *Gaussian distribution* [19, 45]. The former case is often dealt with using the *possible worlds semantics* [46], while the latter one depends on properties of specific probability density functions. Our work falls into the latter category.

There have been diverse studies on adapting data management and mining activities for uncertain data. Several probabilistic database management systems have been proposed, including *Trio* [11], *MayBMS* [12], and *Orion* [13]. In the area of spatio-temporal databases, a large amount of work has been done to support efficient processing of various queries such as *probabilistic range queries* [14, 19, 45], *probabilistic nearest neighbor queries* [21], *probabilistic join queries* [47], and *probabilistic skyline queries* [48]. Their *continuous* versions have also been investigated for moving objects to either query historical records, i.e., trajectories [49, 50], or monitor current activities [51, 52]. In addition, there are other versions incorporated with query types like *threshold* queries [53], *top-k* queries [54], *ranking* queries [55], *aggregate* queries [56], and *reverse* queries [57], and versions in different environments such as *distributed* domains [58], *indoor* space [59], *constrained* space [60], and *road networks* [61]. We show the diagram of our research

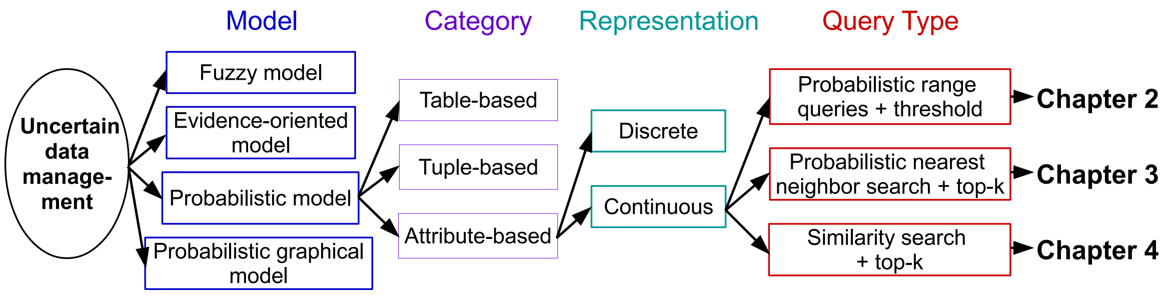


FIGURE 1.4: Research diagram

work in this thesis in Fig. 1.4. Our first work in Chapter 2 combines probabilistic range queries with the threshold query predicate, and our second work in Chapter 3 integrates probabilistic nearest neighbor search with top-k queries. On the other hand, our third work in Chapter 4 considers both similarity search and top-k queries.

Uncertain data management has also been studied in other fields such as *data integration* [62], *data streams* [63], *XML data* [64], *graph data* [65], *sensor networks* [66] and *biological images* [67]. Moreover, mining uncertain data has received much attention from multiple research communities including database, data mining, and information retrieval. A number of algorithms for *clustering* uncertain data [68, 69], *outlier detection* [70] and *frequent pattern mining* [71] from uncertain data are proposed.

1.4 Thesis Organization

The rest of the thesis is organized as follows. In Chapter 2, we introduce the probabilistic range querying over Gaussian objects. Then we present the k-expected nearest neighbor search over Gaussian objects in Chapter 3. Chapter 4 describes the top-k similarity search over Gaussian distributions based on KL-divergence. Finally, in Chapter 5, we conclude our studies and present future directions.

Chapter 2

Probabilistic Range Querying over Gaussian Objects

2.1 Introduction

In the field of uncertain data management, *probabilistic range query* is an important problem for processing uncertain data in real-world applications such as mobile robotics, location-based services, and sensor networks. A probabilistic range query returns all the data objects that appear within the given search region with probabilities no less than a given probability threshold.

For instance, consider a self-navigated mobile robot moving in a wireless environment. The robot builds a map of the environment by observing nearby landmarks via devices such as sonars and laser range finders. Due to the inherent limitation brought about by sensor accuracy and signal noises, the location information acquired from measuring devices is not always precise. At the same time, the robot also conducts probabilistic localization [6] to estimate its own location autonomously by integrating its movement history and the landmark information. This may cause impreciseness in the location of the robot, too. In consequence, probabilistic queries have evolved to tackle such impreciseness as "find landmarks that are located within 5 meters from my current location with probabilities at least 80%".

Typically for such applications, uncertain objects are stored in the database and associated with probability distributions. Multi-dimensional *Gaussian distribution* is one of the commonly used probability distributions for such a purpose. It is widely adopted in statistics, pattern recognition [16], and localization in robotics [6].

In this work we study the case where the locations of data objects are uncertain, whereas the location of the query object is either exact or uncertain. Specifically, data objects are described by Gaussian distributions with different parameters to indicate their respective uncertainty. A query object can be either a certain point in the multi-dimensional space or an uncertain location represented by a multi-dimensional Gaussian distribution. We solve the probabilistic range query problem according to the above setup.

There have been solutions to probabilistic range queries that can handle uncertain data represented by Gaussian distributions, yet based on specific assumptions. For example, U-tree [25] assumes that each uncertain object exists within a *pre-defined uncertainty region*. It constructs an index structure for all the objects based on this region to reduce the number of candidates that require expensive numerical integration. However, for some application scenarios it is not easy to decide a suitable extent of the uncertainty region for a real-world object. Gauss-tree [26] assumes that the Gaussian distribution must be *independent* in each dimension. When these assumptions are violated, the solutions no longer work. In this work, we solve these problems with generic Gaussian distributions without any of these assumptions, i.e., an object can locate in an infinite space as opposed to U-tree, and have correlations between dimensions as opposed to Gauss-tree.

A straightforward approach to this problem is to compute the *appearance probability* [14] for each data object and output it if this probability is no less than the threshold. However, the probability computation usually requires costly numerical integration for the accurate result [25], rendering it prohibitively expensive to compute for all the data objects and check if the query constraint is satisfied. Thus, such computations should be reduced as much as possible.

To this end, we propose filtering techniques to generate a set of candidate data objects and compute integrations only for these candidates. Equipped with the filtering techniques, an R-tree-based indexing method is proposed to accelerate query processing. The index structure is inspired by the idea of TPR-tree [72], in which the minimum bounding boxes (MBBs) vary with time. The difference is that in our index a parent MBB not only varies with the probability threshold but also tightly encloses all the child MBBs.

In our preliminary work [19], we proposed query processing algorithms for probabilistic range queries, assuming that *only the location of the query object* is uncertain and represented by a Gaussian distribution, but data objects are certain multi-dimensional points. An R-tree can be used to manage these certain data points to process queries, which is different from the situation here. In this work, we extend the uncertainty to data objects and propose novel solutions. We develop several effective filtering techniques by analyzing properties of Gaussian distribution thoroughly and design a non-trivial indexing method that can handle multiple online queries.

A precedent report of this work has appeared in [73]. The approach proposed in [73] approximates the Gaussian distribution with an upper-bounding function. An R-tree-like hierarchical index structure was proposed and an exponential summary function was defined to cover multiple upper-bounding functions. Nevertheless, the summary function is so sensitive to the child functions that it will become drastically large if the bounded Gaussian distributions are sparsely distributed in the space or one of them has large variances, leading to loose bounding in the index structure and weak filtering power. This work is an extension of our previous work [45]. We have extended the algorithms and conducted additional experiments.

Our contributions are summarized as follows:

1. We formalize two types of probabilistic range queries with respect to the query object: a certain point and an uncertain location represented by a Gaussian distribution, while data objects are represented by Gaussian distributions with different parameters.

2. For the two types of queries, we propose several effective filtering techniques to identify promising data objects and prune unpromising ones.

3. We design a novel R-tree-based index structure to support probabilistic range queries on Gaussian objects.

4. We demonstrate the efficiency of our approach through a comprehensive experimental performance study using real datasets.

The rest of this chapter is organized as follows. Section 2.2 defines our problem. We present our filtering strategies in Section 2.3. Section 2.4 describes our index structure. Experimental results and analyses are covered by Section 2.5. Section 2.6 reviews related work. Section 2.7 concludes this chapter.

2.2 Problem Definition

In this section, we first define Gaussian objects, and then define probabilistic range queries using two types of query objects: certain point objects and uncertain Gaussian objects.

2.2.1 Gaussian Objects

The Gaussian distribution, also known as normal distribution, is a continuous probability distribution defined by a bell-shaped probability density function in the one-dimensional case. In this work, we assume that data objects are modeled by Gaussian distributions in a d-dimensional space. A point $\boldsymbol{x}$ is in a d-dimensional numerical space, namely, $\boldsymbol{x} = (x^1, .., x^d)^t$.

Definition 2.1 (Gaussian objects)**.** A Gaussian object o is represented by its possible locations (points) and the probability density it appears at each location. Formally, the probability density that o is located at $\boldsymbol{x}_o$ is captured by a d-dimensional Gaussian probability density function

$$p_o(\boldsymbol{x}_o) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}_o|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}_o - \boldsymbol{\mu}_o)^T \boldsymbol{\Sigma}_o^{-1}(\boldsymbol{x}_o - \boldsymbol{\mu}_o)\right]. \tag{2.1}$$

$\boldsymbol{\mu}_o$ is the mean location (center) of o. $\boldsymbol{\Sigma}_o$ is a $d \times d$ covariance matrix. $|\boldsymbol{\Sigma}_o|$ and $\boldsymbol{\Sigma}_o^{-1}$ are the determinant and the inverse of $\boldsymbol{\Sigma}_o$, respectively. $\boldsymbol{x}^T$ denotes the transposition of the vector $\boldsymbol{x}$.

2.2.2 Probabilistic Range Queries on Gaussian Objects

Given a database of Gaussian objects $\mathcal{D}$, a query object q, a distance threshold δ, and a probability threshold θ, a *probabilistic range query (PRQ) on Gaussian objects* retrieves all the data objects $o \in \mathcal{D}$ such that the distance between o and q is no more than δ with probabilities no less than θ. In this work, we consider two types of query objects for q:

1. The query object is a point object, namely,

$$\boldsymbol{q} = (x_q^1, x_q^2, \ldots, x_q^d)^t.$$

2. The query object is a Gaussian object, namely,

$$p_q(\boldsymbol{x}_q) = \frac{1}{(2\pi)^{\frac{d}{2}}|\boldsymbol{\Sigma}_q|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(\boldsymbol{x}_q - \boldsymbol{\mu}_q)^t \boldsymbol{\Sigma}_q^{-1}(\boldsymbol{x}_q - \boldsymbol{\mu}_q)\right].$$

The *probabilistic range query with a point query object* (PRQ-P) is formally defined as

$$\text{PRQ-P}(\mathcal{D}, q, \delta, \theta) = \{o \mid o \in \mathcal{D}, \Pr(||\boldsymbol{x}_o - \boldsymbol{q}|| \leq \delta) \geq \theta\},$$

where $||\boldsymbol{x}_o - \boldsymbol{q}||$ represents the Euclidean distance between $\boldsymbol{x}_o$ and $\boldsymbol{q}$. We call the region consisting of the points with distances no more than δ from the query object *the query region*, *QR* for short. $\Pr(||\boldsymbol{x}_o - \boldsymbol{q}|| \leq \delta)$, the probability that o lies within *QR*, is computed by

$$\Pr(||\boldsymbol{x}_o - \boldsymbol{q}|| \leq \delta) = \int \chi_\delta(\boldsymbol{x}_o, \boldsymbol{q}) \cdot p_o(\boldsymbol{x}_o) d\boldsymbol{x}_o, \tag{2.2}$$

where

$$\chi_\delta(\boldsymbol{x}_o, \boldsymbol{q}) = \begin{cases} 1, & ||\boldsymbol{x}_o - \boldsymbol{q}|| \leq \delta; \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

The integration in Eq. (2.2) is not in a closed-form and hence cannot be computed directly. Numerical solutions such as Monte Carlo methods can be employed to evaluate the probability. We use the *importance sampling* [74] in the interest of efficiency. Specifically, we generate $\boldsymbol{x}_o$ as per the probability function $p_o(\boldsymbol{x_o})$, and increment the count when

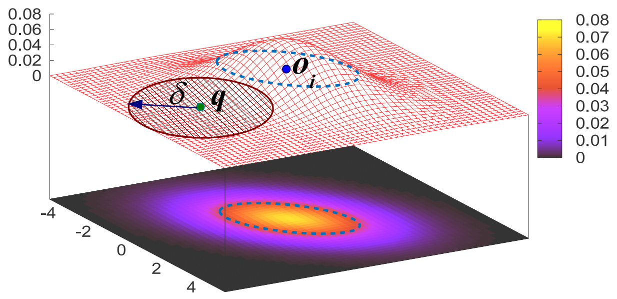


FIGURE 2.1: PRQ-P query

Eq. (2.3) is satisfied. Finally, the probability can be obtained by dividing the count by the number of samples generated. Generally speaking, however, Monte Carlo methods are accurate only if the number of samples is sufficiently large (at the order of 10^6) [25]. Therefore, the probability computation induces expensive runtime cost.

Figure 2.1 illustrates a PRQ-P query in a 2-dimensional space. The Gaussian object o exists in the space with decreasing probability densities as it spreads from the center $\boldsymbol{\mu}_o$. A PRQ-P query finds the Gaussian objects located in the proximity of the query point with a required probability. Computing the probability using Eq. (2.2) corresponds to integrating the probability density function of o within the shaded area around q.

Similar to PRQ-P, the *probabilistic range query with a Gaussian query object* (PRQ-G) is defined as

$$\text{PRQ-G}(\mathcal{D}, q, \delta, \theta) = \{o \mid o \in \mathcal{D}, \Pr(\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta) \geq \theta\},$$

where $\Pr(\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta)$ is computed by

$$\begin{aligned} &\Pr(\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta) \\ &= \iint \chi_\delta(\boldsymbol{x}_o, \boldsymbol{x}_q) \cdot p_o(\boldsymbol{x}_o) \cdot p_q(\boldsymbol{x}_q) d\boldsymbol{x}_o d\boldsymbol{x}_q, \end{aligned} \tag{2.4}$$

where

$$\chi_\delta(\boldsymbol{x}_o, \boldsymbol{x}_q) = \begin{cases} 1, & \|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta; \\ 0, & \text{otherwise.} \end{cases}$$

To compute the integration in Eq. (2.4), although we can simply generate random numbers for the two Gaussian distributions $p_o(\boldsymbol{x}_o)$ and $p_q(\boldsymbol{x}_q)$, respectively, a more efficient method is shown in [73]. It constructs a $2d$-dimensional Gaussian distribution by combining the two d-dimensional Gaussian distributions. Then the probability can be computed by running the Monte Carlo method.

2.3 Filtering Based on Approximated Region

A naïve algorithm to answer PRQ-P or PRQ-G queries is to pair the query object with each data object and perform integration check with either Eq. (2.2) or Eq. (2.4). However, due to the overhead of the integration, the algorithm becomes prohibitively expensive for large datasets. So we develop our approach based on the filter-and-refine paradigm, i.e., to obtain a set of candidate objects and then compute the integration only for the candidates.

In this section, we first introduce the notion of ρ-region that leverages the two thresholds δ and θ, and then propose the ρ-region-based filtering techniques to handle PRQ-P and PRQ-G queries.

2.3.1 ρ-Region

Definition 2.2 (ρ-region)**.** Consider a Gaussian object o and the integration of its probability density function $p_o(\boldsymbol{x}_o)$ over an ellipsoidal region $(\boldsymbol{x}_o - \boldsymbol{\mu}_o)^t\boldsymbol{\Sigma}_o^{-1}(\boldsymbol{x}_o - \boldsymbol{\mu}_o) \le r^2$. Let r_ρ be the value of r within which the integration equals ρ:

$$\int_{(\boldsymbol{x}_o-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_o^{-1}(\boldsymbol{x}_o-\boldsymbol{\mu}_o)\le r_\rho^2} p_o(\boldsymbol{x}_o)d\boldsymbol{x}_o = \rho.$$

We call the ellipsoidal region

$$(\boldsymbol{x}_o - \boldsymbol{\mu}_o)^t\boldsymbol{\Sigma}_o^{-1}(\boldsymbol{x}_o - \boldsymbol{\mu}_o) \le r_\rho^2$$

the ρ*-region* of o.

In Fig. 2.1, the area encompassed by the dotted ellipsoidal curve shows a ρ-region. Because the probability density of a Gaussian distribution decreases as we move away from the center of the object, if the query object is distant enough from the center, its probability within QR will not reach θ. In other words, it is possible to determine whether a data object can satisfy the query condition by deriving a suitable ρ-region with θ (the method is introduced in Section 2.3.3 and 2.3.4) and examining whether its ρ-region intersects QR.

To compute r_ρ with a given ρ, we borrow the approach proposed in our previous work [19]. It transforms the integration over an ellipsoidal region to an integration over a d-dimensional spherical region. By assigning $\boldsymbol{\mu}_o = \mathbf{0}$ and $\mathbf{\Sigma}_o = \mathbf{I}$ in Eq. (2.1), we have the *normalized Gaussian distribution*

$$p_{\mathrm{norm}}(\boldsymbol{x}) = \mathcal{N}(\mathbf{0}, \mathbf{I}) = \frac{1}{(2\pi)^{d/2}} \exp\left[-\frac{1}{2}||\boldsymbol{x}||^2\right].$$

Based on this probability density function, the following property can be derived.

Property 1. [19] Consider integration of $p_{\mathrm{norm}}(\boldsymbol{x})$ over $||\boldsymbol{x}||^2 \leq r^2$. For a given ρ $(0 < \rho < 1)$, let $\tilde{r}_\rho$ be the radius within which the integration becomes ρ:

$$\int_{||\boldsymbol{x}||^2 \leq \tilde{r}_\rho^2} p_{\mathrm{norm}}(\boldsymbol{x}) d\boldsymbol{x} = \rho.$$

Then $r_\rho = \tilde{r}_\rho$ holds.

The preceding property indicates that we can compute $\tilde{r}_\rho$ and hence r_ρ for a given ρ value. Therefore, we can construct a (ρ, r_ρ)-table offline (numerical integration is necessary) and obtain the ρ-region by looking up the corresponding r_ρ from this table. If there is no matched entry for a given ρ, we conservatively return the corresponding r_ρ of the smallest value greater than ρ, so the correctness of the result can be guaranteed.

The ellipsoidal shape of a ρ-region renders it difficult to quickly examine whether the ρ-region intersects QR as well as develop an indexing scheme based on prevalent spatial indexes such as R-tree. Hence we will study the minimum bounding box (MBB) which tightly bounds the ρ-region.

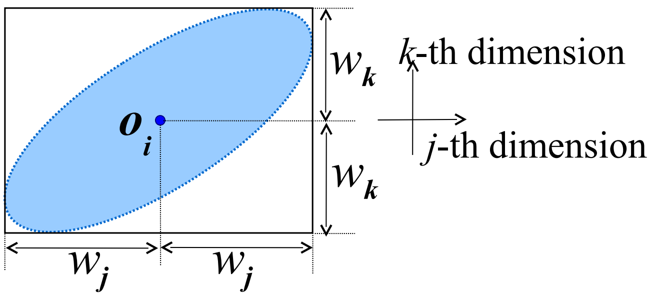


FIGURE 2.2: MBB of ρ-Region

2.3.2 Minimum Bounding Box of ρ-Region

Figure 2.2 shows the MBB of a ρ-region of a 2-dimensional Gaussian object o. Let w_j denote the length of its edge along the j-th dimension. The following property holds [19].

Property 2. The value of w_j ($j = 1, \ldots, d$) is given as

$$w_j = \sigma_j r_\rho, \tag{2.5}$$

where σ_j corresponds to the standard deviation in the j-th dimension

$$\sigma_j = \sqrt{(\mathbf{\Sigma}_o)_{jj}},$$

where $(\mathbf{\Sigma}_o)_{jj}$ represents the (j, j)-th element of $\mathbf{\Sigma}_o$.

For a data object o, since σ_j can be calculated from the covariance matrix $\mathbf{\Sigma}_o$, the scale of the MBB is determined uniquely by r_ρ, and hence ρ. Consequently, in order to establish the filtering conditions utilizing the MBBs, it is essential to explore the relation between ρ and the probability threshold θ. Next we will present our filtering techniques for PRQ-P and PRQ-G, respectively.

2.3.3 Filtering Policies for PRQ-P Queries

Our filtering policies to process PRQ-P queries are divided into two cases: $\theta < 0.5$ and $\theta \geq 0.5$.

Case 1 ($\theta < 0.5$)**:** Consider the four data objects o_1, o_2, o_3, o_4 shown in Fig. 2.3(a). $bb_i(\rho)$ denotes the MBB of o_i's ρ-region. First, let's consider o_4. Since the probability that o_4 is located inside its ρ-region is ρ, the probability of being outside the ρ-region's MBB, is definitely less than $1-\rho$. Furthermore, given the line symmetry of the Gaussian distribution, the probability that o_4 exists inside QR is at most $(1-\rho)/2$. Hence, if $\rho = 1 - 2\theta$, and $bb_4(\rho)$ and QR do not overlap, the probability that o_4 lies within QR must be less than θ. Then, for objects o_1, o_2 and o_3, we check and find their MBBs intersect QR, so they become candidates. In summary, when $\theta < 0.5$, a data object is a candidate only if its $bb_i(1-2\theta)$ intersects QR.

Case 2 ($\theta \geq 0.5$)**:** We show our idea in Fig. 2.3(b). For the probability that a data object exists inside QR to reach θ, its mean location should lie within QR. Thus, o_2 and o_4 can be pruned, whereas o_1 and o_3 are considered as candidates.

Moreover, for all candidates, let $\rho' = \theta$ and compute their $bb_i(\rho')$'s. If QR fully contains a $bb_i(\rho')$ (e.g., o_3), the probability that this object lies within QR is definitely greater than θ. We validate it as a result without computing the numerical integration.

Summarizing the two cases, the filtering condition for PRQ-P is: (1) $\theta < 0.5$, $bb_i(\rho)$ ($\rho = 1 - 2\theta$) intersects QR for pruning, and $bb_i(\rho')$ ($\rho' = \theta$) is contained by QR for validation. (2) $\theta \geq 0.5$, $\|\boldsymbol{\mu}_o - \boldsymbol{q}\| < \delta$ for pruning, and $bb_i(\rho')$ ($\rho' = \theta$) is contained by QR for validation.

2.3.4 Filtering Policies for PRQ-G Queries

For PRQ-G queries, since both the query object q and the data object o_i are Gaussian distributions, we obtain both of their MBBs of ρ-regions. We also consider two cases for filtering: $\theta < 0.75$ and $\theta \geq 0.75$.

Case 1 ($\theta < 0.75$)**:** As shown in Fig. 2.3(c), assume q and o_i are independent in the space. When the minimum distance between the two MBBs is exactly δ, the maximal probability of $\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta$ is given by the following lemma (the proof is in A.1):

Lemma 2.3. *If $MinDist(bb_i(\rho), bb_q(\rho)) \geq \delta$, $Pr(\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta) < (3 - 2\rho - \rho^2)/4$.*

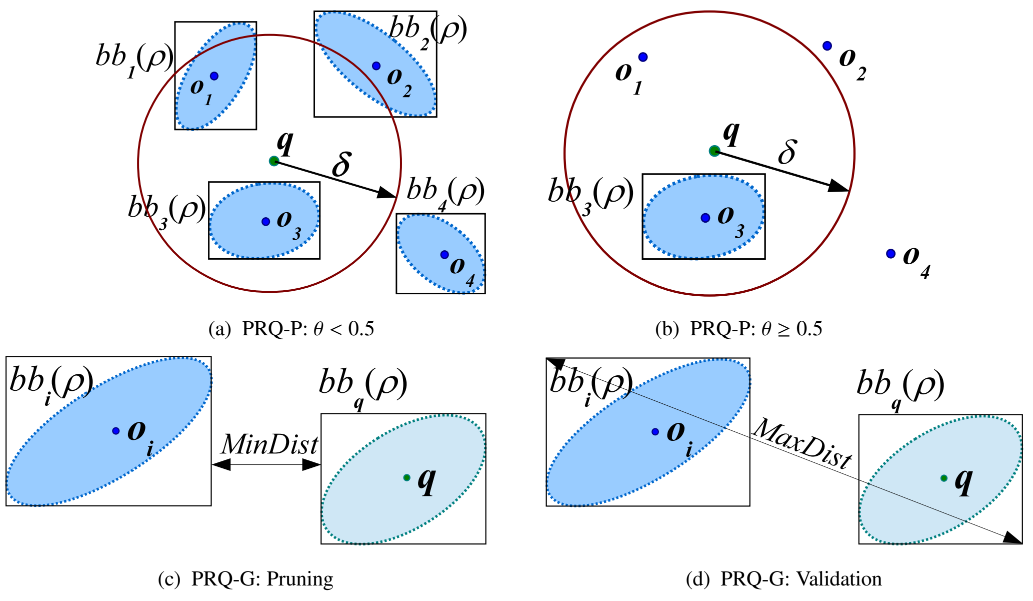


FIGURE 2.3: Filtering policies for PRQ-P and PRQ-G

Let $(3 - 2\rho - \rho^2)/4 = \theta$, i.e., $\rho = 2\sqrt{1-\theta} - 1$. o_i can be excluded from the candidate set if the minimum distance between $bb_i(\rho)$ and $bb_q(\rho)$ is no less than δ.

Lemma 2.4. *If* $||\boldsymbol{\mu}_o - \boldsymbol{\mu}_q|| \geq \delta$, $Pr(||\boldsymbol{x}_o - \boldsymbol{x}_q|| \leq \delta) < 0.75$.

Case 2 ($\theta \geq 0.75$)**:** Based on Lemma 2.4 (the proof is in A.2), an object can be pruned if the distance between its center and that of the query object is no less than δ.

Moreover, let $\rho' = \sqrt{\theta}$, if the maximum distance of $bb_i(\rho')$ and $bb_q(\rho')$ is less than δ, as shown in Fig. 2.3(d), the probability of $||\boldsymbol{x}_o - \boldsymbol{x}_q|| \leq \delta$ is greater than ρ'^2, i.e., θ. In this case, we validate it as a result without computing the exact probability by numerical integration.

The filtering condition for PRQ-G is summarized as: (1) $\theta < 0.75$, $MinDist(bb_i(\rho), bb_q(\rho)) < \delta$ ($\rho = 2\sqrt{1-\theta} - 1$) for pruning, and $MaxDist(bb_i(\rho'), bb_q(\rho')) < \delta$ ($\rho' = \sqrt{\theta}$) for validation. (2) $\theta \geq 0.75$, $||\boldsymbol{\mu}_o - \boldsymbol{\mu}_q|| < \delta$ for pruning, and $MaxDist(bb_i(\rho'), bb_q(\rho')) < \delta$ ($\rho' = \sqrt{\theta}$) for validation.

2.4 Indexing Data Objects

2.4.1 The Index Structure

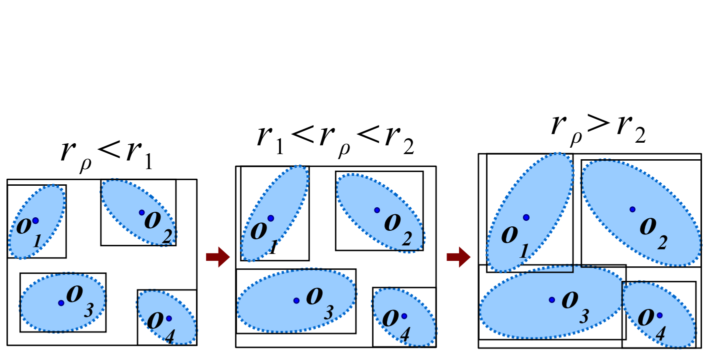
The filtering conditions introduced in the previous section need to know the value of θ and hence ρ to generate candidates. In order not to scan all the data objects and compute the MBBs of the ρ-regions on the fly with the given θ, an immediate solution is to index the MBBs for a sufficiently large $\rho_{\max}$. Because the MBB with a larger ρ always subsumes the one with a smaller ρ, it can support all the queries if the ρ values computed from θ satisfy the condition $\rho \leq \rho_{\max}$. However, the efficiency of the index is compromised for small ρ values. This method serves as a baseline algorithm (we use an R-tree to index MBBs and name it FR-tree), and will be compared in the experiment with the indexing technique we are going to present.

Inspired by the TPR-tree [72], we propose an R-tree-based index structure which stores the MBBs in a parametric fashion. It works for arbitrary probability thresholds and range thresholds, and there is no need to assume the two thresholds are given prior to index construction. The MBBs can be dynamically computed as we traverse the index. Furthermore, the MBB of a node (at both leaf and non-leaf levels) tightly encloses all its children's MBBs regardless of the θ value, as opposed to the TPR-tree within which all child MBBs are bounded in a loose manner.

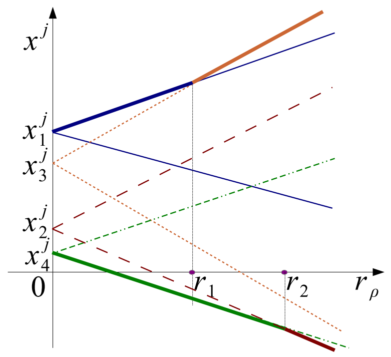
Our index is a balanced, multi-way tree organized in the structure of an R-tree. Each entry in a leaf node contains a data object in the form of $o_i = (id_i, \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where id_i is the data object id, $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the mean location and the covariance matrix of the Gaussian object, respectively. In a non-leaf node, each entry has a pointer to a child node and an MBB enclosing all the MBBs within the child node.

Consider an object o_i with mean location $(x_i^1, \ldots, x_i^d)^t$. Its MBB is a bounding box parameterized with r_ρ (MBB is denoted as the exactly computed bounding box at a specific r_ρ hereafter). From Eq. (2.5), the extent of the bounding box (BB) in the j-th dimension can be represented by

$$[x_i^j - w_i^j, x_i^j + w_i^j] = [x_i^j - \sigma_i^j r_\rho, x_i^j + \sigma_i^j r_\rho]. \tag{2.6}$$



(a) Node BB of Four Child BBs



(b) Left and Right Edges of the BB

FIGURE 2.4: BB with varying ρ

Seeing the BBs grow with r_ρ, in order to tightly bound the BBs of child nodes at every r_ρ, it is necessary to search each dimension for the leftmost and the rightmost BBs under varying ρ. We illustrate this problem in Fig. 2.4(a). It shows the changing BB that encloses the BBs of four 2-dimensional objects' ρ-regions as r_ρ increases. When r_ρ is less than r_1, the left edge is determined by o_1, and it becomes o_3 when r_ρ exceeds r_1. The right bound is determined by o_4 when $r_\rho < r_2$, and o_2 otherwise.

Figure 2.4(b) shows how the four BBs change horizontally with r_ρ. For each object, a pair of symmetrical lines describe the left and the right coordinations of its BB. The lines have different slopes due to the difference in their standard deviations. The bold polylines illustrate the left and right coordinations of the outer enclosing BB. Hence, the problem becomes how to find the bold polylines. To this end, a BB can be represented by several segments with respect to r_ρ.

We store in the index the j-th dimension of a BB in the form of $(\langle x_1^j, \sigma_1^j, r_1\rangle, \ldots, \langle x_k^j, \sigma_k^j, +\infty\rangle)$. For example, for the four objects in Fig. 2.4(a), the left and the right coordinations of the BB are $(\langle x_1^j, \sigma_1^j, r_1\rangle, \langle x_3^j, \sigma_3^j, +\infty\rangle)$ and $(\langle x_4^j, \sigma_4^j, r_2\rangle, \langle x_2^j, \sigma_2^j, +\infty\rangle)$, respectively.[1]

We can find all the segments in the j-th dimension through a sort on the coordinations first and then a linear scan from the object whose standard deviation has a larger value in the j-th dimension. The time complexity is $O(n \log n)$, where n is the number of its child nodes. The number of segments in a BB is at most n.

[1] Our experiments show the average number of segments in a BB is 2–3. In case of many segments, we allow users to specify a query probability threshold range $[\theta_{min}, \theta_{max}]$ to reduce the number of segments and hence the index size.

We note the difference between our index and TPR-tree: (1) The bounding boxes of TPR-tree change towards one direction in a rate (velocity), while our bounding boxes change towards two opposite directions symmetrically with r_ρ. (2) The bounding boxes of TPR-tree are tight only if an update occurs, while our bounding boxes are always tight.

2.4.2 Query Processing with the Index

Algorithm 1 and Algorithm 2 show the procedures of query processing of PRQ-P (the algorithms for PRQ-G are similar and thus omitted due to the space limitation). To process a query, θ is converted to ρ and ρ', and then r_ρ and $r_{\rho'}$ with the pre-computed (ρ, r_ρ) table (line 1–2). A first-in-first-out queue Q is employed to maintain the nodes we are going to expand. Starting with the root node, we compare QR (MBB of q for PRQ-G) with the node MBB, and check the filtering condition. To compute a node MBB, given r_ρ, we scan the stored jth-dimension of its BB and find α such that $r_{\alpha-1} \leq r_\rho < r_\alpha$. Then the extent of its MBB in the j-th dimension can be computed through Eq. (2.6) using x_α, σ_α, and r_ρ.

At the non-leaf level (line 7–18), if a node MBB at $r_{\rho'}$ satisfies the validation condition, i.e., it is contained by QR (for PRQ-G, the maximum distance of two MBBs is less than δ), we retrieve all the objects within this node and push them into the result set directly; otherwise, we probe each child entry. If $\theta < 0.5$ (0.75 for PRQ-G), we check whether each child MBB at r_ρ intersects QR (for PRQ-G, whether the minimum distance between each child MBB and the MBB of q is less than δ). On the other hand, when $\theta \geq 0.5$ (0.75 for PRQ-G), we obtain the MBB which bounds leftmost and rightmost means in each dimension, and use this more compact MBB as the new child MBB for filtering checking for both PRQ-P and PRQ-G.

At the leaf level (line 19–30), we also examine whether a node MBB at $r_{\rho'}$ satisfies the validation condition first, so as to push its child entries into the result set as early as possible and reduce the processing cost. If not, each child entry of this node is processed as discussed in Section 2.3.3 (for PRQ-G, Section 2.3.4). When $\theta < 0.5$ (0.75 for PRQ-G), for each child entry, if its MBB at r_ρ intersects QR (for PRQ-G, the minimum distance between each child MBB and the MBB of q is less than δ), we identify this data object

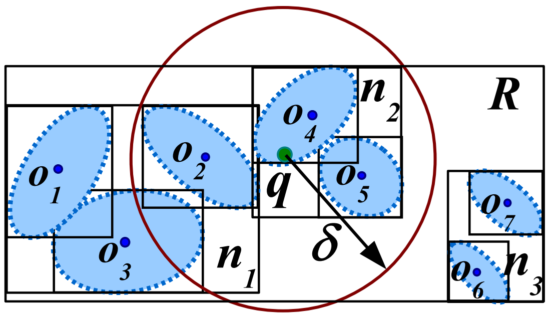


FIGURE 2.5: An illustration of PRQ-P query processing

using Algorithm 2. We check whether the validation condition is satisfied. The qualified data objects are inserted directly into the result set. Other data objects are regarded as candidates for further verification through integration. When $\theta \geq 0.5$ (0.75 for PRQ-G), for each child entry, if its mean location is inside QR, it is checked by Algorithm 2 in the same way.

Figure 2.5 shows a query processing example for PRQ-P with $\theta = 0.3$. $r_{0.4}$ and $r_{0.3}$ are obtained from the (ρ, r_ρ) table. MBBs of all nodes in Fig. 2.5 are at $r_{0.4}$. At first, the root R is popped from Q. Since the MBB of R cannot satisfy the validation condition, we have to check its child nodes n_1, n_2 and n_3. Only n_1 and n_2 are pushed into Q because the MBB of n_3 does not intersect QR. Then we continue with the next level. For n_1, its MBB at $r_{0.4}$ is not contained by QR, so its child entries cannot be all validated as results. Since its MBB at $r_{0.3}$ intersects QR, we probe into its child entries and find that o_1 can be pruned. Then o_2 is validated as a result object and o_3 becomes a candidate further processed through integration. For n_2, since its MBB at $r_{0.3}$ is completely within QR, its entries o_4 and o_5 are inserted into the result set directly without referring to their MBBs.

2.5 Experiments

2.5.1 Experimental Setup

Three real datasets are used in our experiments. MG and LB are two 2-dimensional datasets of Montgomery and Long Beach road networks (39K and 52K, respectively)[2].

[2]http://www.census.gov/geo/www/tiger

Algorithm 1 PRQ-P($\boldsymbol{q}, \delta, \theta$)

```
1: $\rho \leftarrow 1 - 2\theta, \rho' \leftarrow \theta$
2: $r_\rho \leftarrow TableLookup(\rho), r_{\rho'} \leftarrow TableLookup(\rho')$
3: $QR \leftarrow QueryRegion(\boldsymbol{q}, \delta), ResultSet = \emptyset, CandidateSet = \emptyset$
4: $Q \leftarrow Root, N \leftarrow 0$
5: while $Q \neq \emptyset$ do
6:     $N \leftarrow Q.pop()$
7:     if $N$ is an internal node then                          ▷ at the non-leaf level
8:         if $bb_N(r_{\rho'})$ IsContainedBy $QR$ then
9:             Insert all the objects within $N$ into $ResultSet$
10:        else
11:            if $\theta < 0.5$ then
12:                for each child $N_i$ in $N$ do
13:                    if $bb_{N_i}(r_\rho)$ Intersects $QR$ then
14:                        Push $N_i$ into $Q$
15:            else                                             ▷ $\theta \geq 0.5$
16:                for each child $N_i$ in $N$ do
17:                    if $meanbb_{N_i}$ Intersects $QR$ then
18:                        Push $N_i$ into $Q$
19:    else                                                     ▷ at the leaf level
20:        if $bb_N(r_{\rho'})$ IsContainedBy $QR$ then
21:            Insert all the objects within $N_i$ into $ResultSet$
22:        else
23:            if $\theta < 0.5$ then
24:                for each child $N_i$ in $N$ do
25:                    if $bb_i(r_\rho)$ Intersects $QR$ then
26:                        IdentifyObjects($N_i$)
27:            else                                             ▷ $\theta \geq 0.5$
28:                for each child $N_i$ in $N$ do
29:                    if $mean(N_i)$ IsInside $QR$ then
30:                        IdentifyObjects($N_i$)
```

Airport is a 3-dimensional dataset containing latitudes, longitudes and elevations of 41K airports in the world[3]. All datasets are normalized to $[0, 1000]^d$. LB is used by default.

We generate PRQ-P and PRQ-G queries randomly. The probability threshold θ lies within $[0.01, 0.99]$, and the query range δ is chosen from $[10, 100]$ for MG and LB, and $[100, 200]$ for Airport randomly. We also generate covariance matrices for both data Gaussian objects and query Gaussian objects randomly.

[3]http://www.ourairports.com/data

Algorithm 2 IdentifyObjects(N_i)

```
1: if $bb_{N_i}(r_{\rho'})$ IsContainedBy $QR$ then
2:     Insert $N_i$ into $ResultSet$
3: else
4:     $Integral \leftarrow computeIntegral(N_i)$
5:     if $Integral \geq \theta$ then
6:         Insert $N_i$ into $ResultSet$
```

We design two baseline approaches for experimental evaluation. Both of them and our proposed approach return the same exact result. We check their correctness before each comparison. One baseline approach is to sequentially scan the dataset and compute integrations required for obtaining result probabilities. We name it Scan and evaluate our filtering techniques by comparing query processing time and candidate number with it. The other baseline approach indexes the MBBs of the ρ-region with $\rho_{\max} = 0.99$. Because the MBB with a larger ρ always subsumes the one with a smaller ρ, it can support all the queries if the ρ values computed from θ satisfy $\rho \leq \rho_{\max}$. We equip this index with our basic filtering techniques and name it FR-tree (Filtering + R-tree), and evaluate our index structure by comparing filtering time and I/O access with it. Our index is referred to as G-tree.

We implement the index structure by extending the spatial index library SaiL[4] [75]. It is a generic framework that integrates spatial and spatio-temporal index structures and supports user-defined datatypes and customizable spatial queries. We conduct experiments using a PC with Intel Core 2 Duo CPU E8500 (3.16GHz), RAM 4GB, running Fedora 12. We construct an index of all data objects for both PRQ-P and PRQ-G, and store it in the secondary memory.

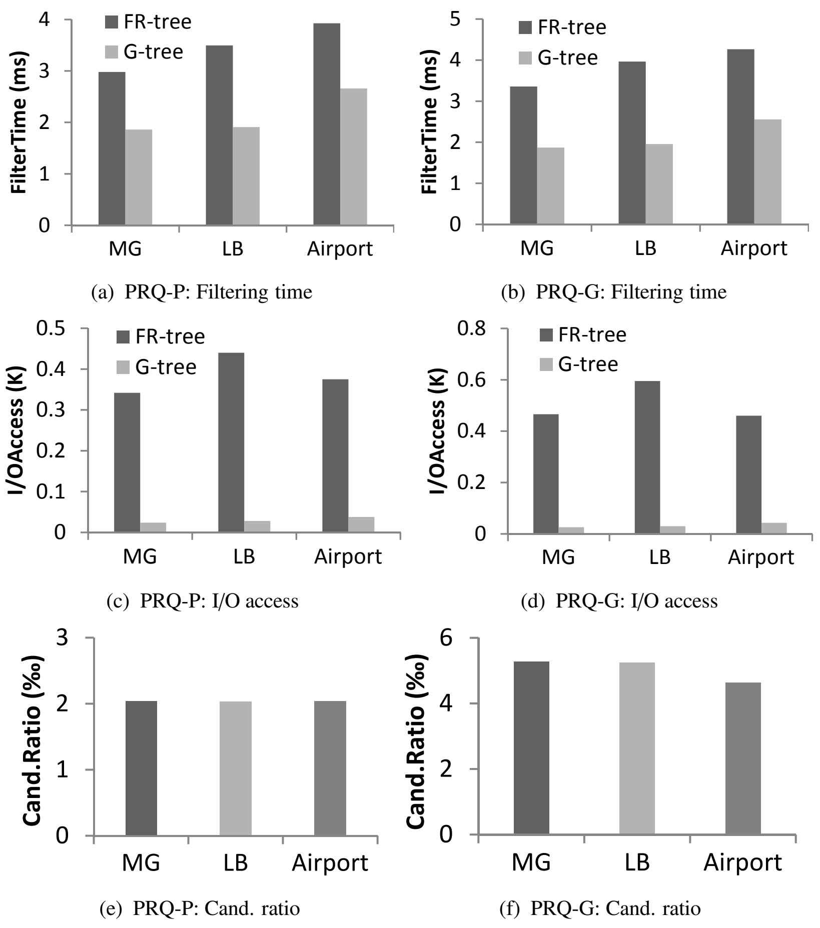
2.5.2 Query Performance Evaluation

The average query response time of 200 queries on LB is shown in Table 2.1. It can be seen that the query response time of Scan is 769 times that of G-tree for PRQ-P (293 for PRQ-G). Another observation is that the time spent on probability integration is almost equal to the overall response time. This indicates that the integration dominates the

[4]http://libspatialindex.github.com/

Query / Algorithm	Overall	Integration
PRQ-P / G-tree	0.157	0.154
PRQ-P / Scan	120.764	120.692
PRQ-G / G-tree	0.809	0.806
PRQ-G / Scan	236.725	236.577

TABLE 2.1: Query response time on LB (seconds)



(a) PRQ-P: Filtering time
(b) PRQ-G: Filtering time
(c) PRQ-P: I/O access
(d) PRQ-G: I/O access
(e) PRQ-P: Cand. ratio
(f) PRQ-G: Cand. ratio

FIGURE 2.6: Filtering and indexing performance

overall query processing and is computationally expensive. Consequently, it is important to reduce candidates which need to perform the integration as much as possible.

Among 50,747 objects in LB, the average candidate number of G-tree is 97 for PRQ-P (269 for PRQ-G). The number of validated objects by integration is 65 for PRQ-P (156 for PRQ-G). So for PRQ-P 67% (58% for PRQ-G) of the candidates identified by our approach are real results. This demonstrates the effectiveness of our proposed filtering techniques. In the sequel, we exclude the integration part from query processing and focus on evaluating the filtering and indexing performance of FR-tree and G-tree.

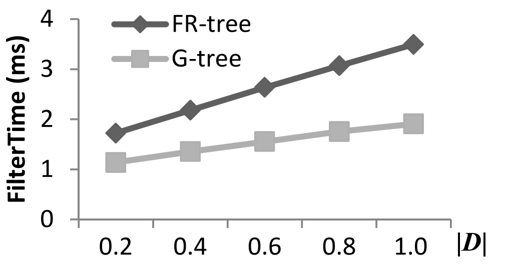
We run the two algorithms to process 10K queries on the three datasets and show the average filtering time and I/O access of PRQ-P (resp. PRQ-G) in Fig. 2.6(a) – 2.6(c) (resp. Fig. 2.6(b) – 2.6(d)). For PRQ-P, the average filtering time of G-tree is 61.6% of that of FR-tree on the three datasets, because the average I/O access of G-tree is 92.2% less than that of FR-tree, though the segmented bounding boxes in G-tree are more complex to process than those in FR-tree. The reduction on PRQ-G is more substantial. The average filtering time of G-tree on MG, LB and Airport is 45% less than that of FR-tree. The I/O access of G-tree of three datasets is 6.5% that of FR-tree.

As a $\rho_{\max}$ is adopted to process queries with any θ, the bounding boxes in FR-tree are very loose. This causes more I/O accesses and increases filtering time. In contrast, since the bounding boxes in G-tree are constructed in a parametric fashion, they can be calculated dynamically for arbitrary θ and hence are very compact. Another interesting observation is that the I/O access almost resembles the candidate number, indicating most I/Os are spent on retrieving objects.

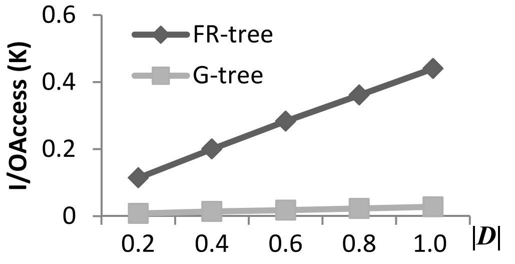
Figure 2.6(e) – 2.6(f) show the candidate ratio of PRQ-P and PRQ-G, which is calculated by dividing the candidate number by the total number of objects. The candidate number of FR-tree and G-tree is the same since we equip FR-tree with our filtering techniques. The candidate ratio is around 2‰ for PRQ-P and 5‰ for PRQ-G on the three datasets. This reveals that only a very small percentage of data objects will become candidates owing to our filtering techniques.

Varying Dataset Size. To evaluate the scalability of our approach, we extract 20%, 40%, 60%, 80% and 100% of LB dataset randomly and show the filtering time and I/O access of two methods in Fig. 2.7(a) – 2.7(b) on PRQ-P queries. The performance on PRQ-G queries reveals a similar trend and hence is omitted due to the space limitation. As the dataset size becomes larger, the filtering time and I/O access of the two methods almost increase linearly. G-tree displays a steady increasing trend and always outperforms FR-tree.

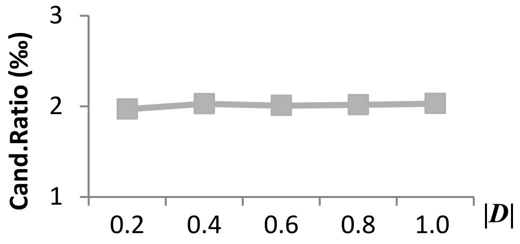
As shown in Fig. 2.7(c) – 2.7(d), in spite of the varying dataset size $|\mathcal{D}|$, the candidate ratio of PRQ-P retains 2‰ and 5.2‰ for PRQ-G, demonstrating the steadiness and scalability of our approach with respect to the dataset size.



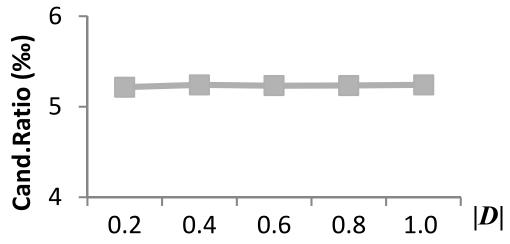
(a) PRQ-P: Filtering time



(b) PRQ-P: I/O access

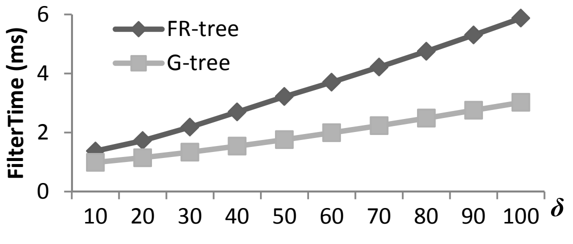


(c) PRQ-P: Candidate ratio

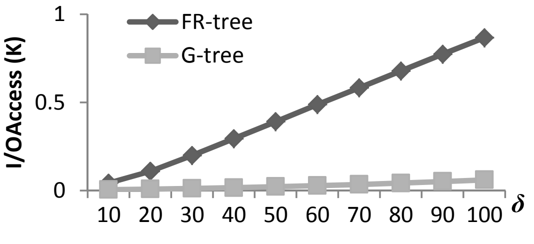


(d) PRQ-G: Candidate ratio

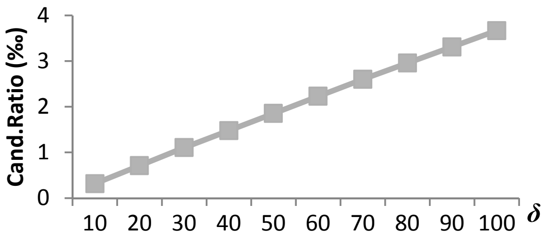
FIGURE 2.7: Varying $|\mathcal{D}|$



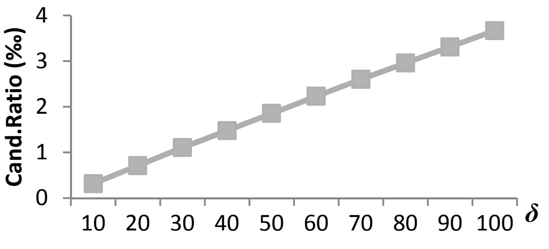
(a) PRQ-P: Filtering time



(b) PRQ-P: I/O access



(c) PRQ-P: Candidate ratio

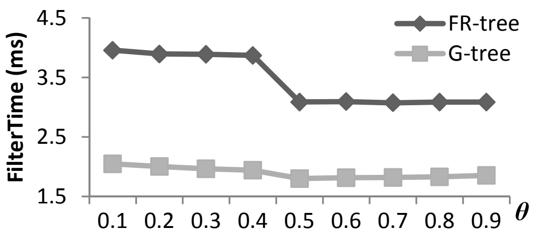


(d) PRQ-G: Candidate ratio

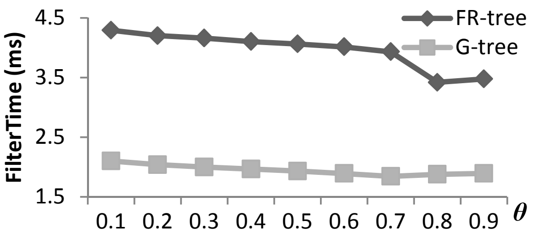
FIGURE 2.8: Varying δ

Varying Query Range. We vary the query range δ from 10 to 100 and show the performance on PRQ-P queries in Fig. 2.8(a) – 2.8(b). The performance on PRQ-G queries is similar and hence omitted. As δ increases, FR-tree consumes much more time and I/O accesses on filtering. In contrast, G-tree exhibits much slower increasing trends. Figure 2.8(c) – 2.8(d) show that the candidate ratio of both PRQ-P and PRQ-G also increases with δ, but for PRQ-P it is only 3.7‰ (9.4‰ for PRQ-G) even if δ achieves 100.

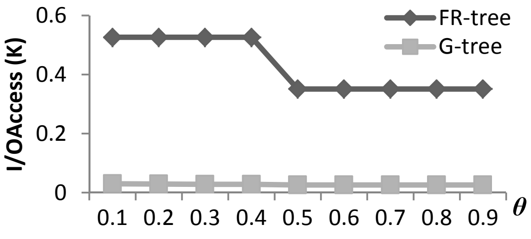
Varying Probability Threshold. We vary θ from 0.1 to 0.9 and show the performance in



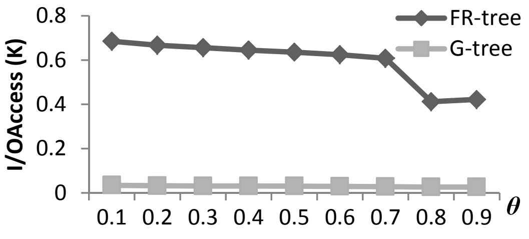
(a) PRQ-P: Filtering time



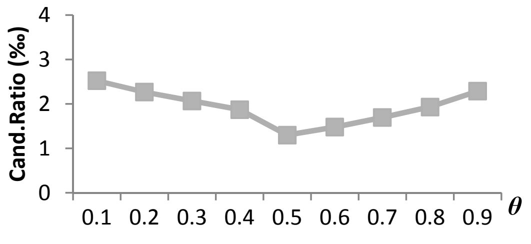
(b) PRQ-G: Filtering time



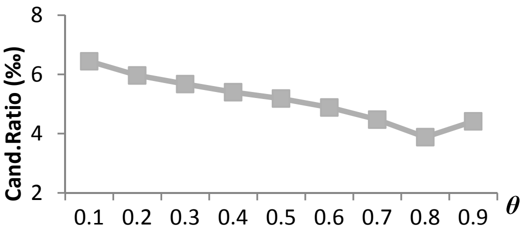
(c) PRQ-P: I/O access



(d) PRQ-G: I/O access



(e) PRQ-P: Candidate ratio



(f) PRQ-G: Candidate ratio

FIGURE 2.9: Varying θ

Fig. 2.9(a) – 2.9(f) for both PRQ-P and PRQ-G queries. For PRQ-P, the filtering time and I/O access of both FR-tree and G-tree decrease gradually with θ when $\theta < 0.5$ (0.75 for PRQ-G). When θ exceeds 0.5 (0.75 for PRQ-G), the filtering time slightly rebounds. This is consistent with our filtering conditions as discussed in Section 2.3.3 and Section 2.3.4. When $\theta < 0.5$ (0.75 for PRQ-G), $\rho = 1 - 2\theta$ for PRQ-P (for PRQ-G, $\rho = 2\sqrt{1-\theta} - 1$ if $\theta < 0.75$), so ρ decreases when θ moves towards larger values, and bounding boxes shrink. As a result, most of non-candidates can be filtered quickly and less I/O accesses are needed, and hence it accelerates filtering.

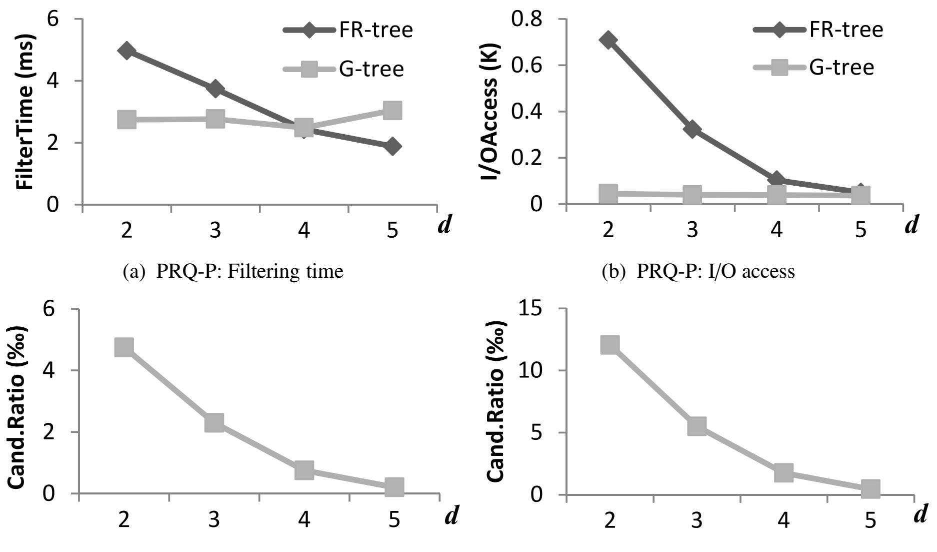
On the contrary, when $\theta \geq 0.5$ (0.75 for PRQ-G), the pruning condition becomes $||\boldsymbol{\mu}_o - \boldsymbol{q}|| < \delta$ for PRQ-P (for PRQ-G, $||\boldsymbol{\mu}_o - \boldsymbol{\mu}_q|| < \delta$). So all the figures have turn points at 0.5 for PRQ-P (0.75 for PRQ-G). At the same time, bounding boxes computed for validation which assigns $\rho' = \theta$ for PRQ-P (for PRQ-G, $\rho' = \sqrt{\theta}$) enlarge as θ increases. So more and more nodes and objects will become candidates, leading to the slightly rising of the filtering time and I/O access. The reason also accounts for the trend of G-tree on candidate ratio in Fig. 2.9(e) – 2.9(f).

Despite the variation of θ, G-tree constantly outperforms FR-tree. In the case of PRQ-P, the filtering time of FR-tree amounts to 1.8 times that of G-tree on average and 15.8 times on average for I/O access. This contrast is more evident on PRQ-G, where the filtering time of FR-tree is 2 times that of G-tree on average and 20.1 times on average for I/O access.

Varying Dimensionality. We also study the impact of dimensionality d using randomly generated synthetic datasets with the size 20K and the query range within $[100, 200]$. Figure 2.10(a) – 2.10(d) show the scalability of FR-tree and G-tree against d for PRQ-P. The figures of filtering time and I/O access for PRQ-G have similar trends and are omitted. As shown in Fig. 2.10(a), the filtering time of FR-tree reduces constantly with increasing d because the object density decreases with d. This can be confirmed by the decreasing trend of I/O access of PRQ-P in Fig. 2.10(b) and the candidate ratio of both PRQ-P and PRQ-G in Fig. 2.10(c) – 2.10(d).

It is also observed that the filtering performance of FR-tree begins to exceed that of G-tree at $d = 5$. The explanation is that, candidates become fewer as object density decreases, and hence the operation of comparing the query region with node bounding boxes dominates the filtering procedure. While FR-tree's MBBs can be obtained directly from the R-tree, G-tree needs to compute the MBBs with the segments in the index. As d increases, the decreasing object density can lower the processing cost. However, the increasing d can also induce more effort in query processing since the bounding box computation needs to be done for more dimensions. The above two factors result in the fluctuating trend of G-tree's filtering time in Fig. 2.10(a).

Index construction. We evaluate the index construction on the Airport dataset. The node capacities of the indexes are selected to optimize query performance for both FR-tree and G-tree. The index size of FR-tree is 6.94MB, and the construction time is 4.3 seconds on average. G-tree has a size of 6.14MB, slightly smaller than FR-tree due to the different entry style of an object from FR-tree. At the leaf level of FR-tree, for each object, besides the bounding box at ρ_{max}, the covariance matrix has to be stored additionally in order to carry out our filtering techniques. In contrast, in G-tree the bounding box of an object is formed by its mean and standard deviation in each dimension. In addition to the



(a) PRQ-P: Filtering time

(b) PRQ-P: I/O access

(c) PRQ-P: Candidate ratio

(d) PRQ-G: Candidate ratio

FIGURE 2.10: Varying *d*

bounding box, only the covariances between dimensions rather than the whole covariance matrix need to be stored. G-tree takes 59.7 seconds on average to build. Although G-tree needs more construction time, considering the superior query performance and the index construction can be done offline, the index construction is in an affordable manner.

2.6 Related Work

Uncertain Data Management. We focus on research work in the area of unertain data management that is closely related to our work. A number of approaches for managing uncertain data have been proposed. Early research primarily focused on queries in a moving object database model [76–79]. The solutions to several types of probabilistic queries were proposed in [44], including probabilistic range queries, where their target is merely the one-dimensional case.

A range query processing method for the case where both data objects and query object are imprecise was proposed in [80]. But they assume that each object exists within a rectangular area. Zheng et al. [81] modeled a fuzzy object by a fuzzy set where each element is characterized by its probability of membership (the sum of all probabilities is not necessarily one). For efficient query processing, they proposed the notion of α-cut, the subset of elements whose probabilities are no less than a user-specified probability threshold α, to filter elements of the fuzzy object. Although we also exploit the idea of filtering region (ρ-region), their rationale of computing the α-cut is different from ours.

Gauss-tree [26] was proposed as an index structure for Gaussian distributions. It assumes all Gaussian distributions are probabilistically *independent* in each dimension. This imposes heavy restriction on the generality of the approach and the overall accuracy of the query result is limited. In [82], Lian et al. proposed a generic framework to tackle the local correlations among uncertain data.

Indexing Uncertain Data for Range Queries. Agarwal et al. [17] presented various indexing structures on uncertain data that support range queries in the one-dimensional case. Tao et al. proposed U-tree [25] to process probabilistic range queries in a multi-dimensional space, where uncertain objects are assumed to follow arbitrary probability distributions within uncertainty regions. Zhang et al. proposed a quadtree-based index called U-Quadtree [83] for range searching on multi-dimensional uncertain data. They mainly focused on representing uncertainty by discrete instances inside a minimum bounding box. The difference from our work is that we take advantage of specific properties of Gaussian distribution and index uncertain objects distributed in an infinite space.

Spatial Data Indexing. The traditional spatial database has been well studied and many indexing methods have been proposed [84–86] to support spatial query processing. R-tree [85] and its extension R*-tree [84], indexing objects by deriving their minimum bounding rectangles (MBRs), are two of the well-known ones. TPR-tree [72] and TPR*-tree [87] were proposed to index moving objects. However, none of these indexes can be applied directly on the Gaussian objects to support the queries studied in this work.

2.7 Summary

In this work, we studied probabilistic range queries over uncertain data. We assumed that the location of the query object is either fixed or follows a multi-dimensional Gaussian distribution. The locations of data objects are represented by Gaussian distributions.

Given these assumptions, we defined two types of probabilistic range queries with respect to the query object. We proposed effective filtering techniques to reduce costly exact probability computation as much as possible. Furthermore, we proposed a novel R-tree-based index structure to expedite query processing. Extensive experiments on real datasets demonstrate the efficiency and effectiveness of our proposed approach.

Chapter 3

k-Expected Nearest Neighbor Search over Gaussian Objects

3.1 Introduction

In recent years, the advances in computing devices and technologies have been calling for the attention of researchers to develop novel solutions for emerging new problems. For instance, the location information obtained from mobile devices such as a mobile phone is usually uncertain due to privacy protection, delays in location update from users, noise, etc. In the area of database, there has been a large amount of work on representing uncertain location information by probabilistic models and proposing efficient solutions for query processing [10]. In particular, *Gaussian distribution* is frequently used to represent this kind of *probabilistic location information* since it is one of the most typical probability distributions, and is widely used in statistics, pattern recognition, and machine learning [15, 16]. For example, our first work in Chapter 2 presents probabilistic range queries over objects represented by Gaussian distributions.

On the other hand, as one of the commonest queries over location information, the distance-based *nearest neighbor search* has applications in numerous fields such as databases, pattern recognition, cluster analysis, and recommendation systems. Given a query point

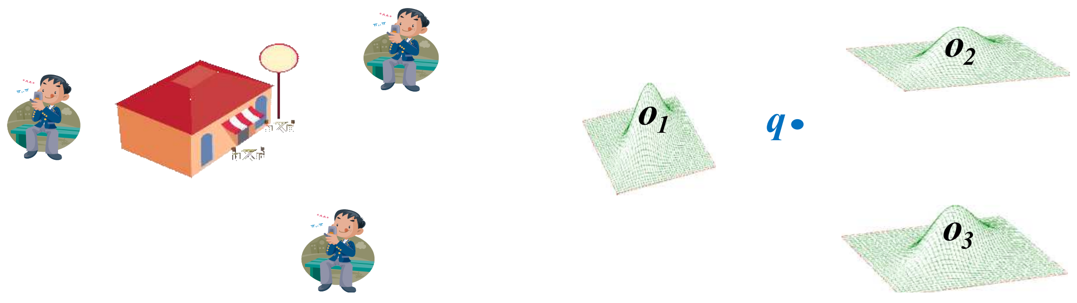


FIGURE 3.1: An example

q, this query searches for closest objects to q in a set of objects. There have been considerable efforts made to extend nearest neighbor search over traditional location information to probabilistic location information. One representative example is the *expected distance* [22, 23], which defines the distance over probabilistic location information. Following this trend, in this work, we represent probabilistic location information by Gaussian distributions and assume that the closeness between each Gaussian object and the query point is measured by their *expected squared Euclidean distance* [23], which is frequently used in many areas such as pattern mining and cluster analysis. Under this setting, we consider the problem of *k-expected nearest neighbor search over Gaussian objects*.

Although there has been a considerable amount of research work on searching over Gaussian distributions [20, 26], to the best of our knowledge, we are the first to consider the problem of k-expected nearest neighbor search over objects represented by Gaussian distributions. Given a database D of objects represented by Gaussian distributions, a query point q, and a constant k, we search D for the top-k objects having smallest distances with q. As shown in Fig. 3.1, an application example is to find nearby mobile users to a given query location such as a shop or a restaurant. Since the location of a mobile user is uncertain, it is a common practice to represent uncertain locations using Gaussian distributions in the area of spatial databases [20]. The restaurant or shop may consider sending coupons or notifications to a number of nearest users for promotion.

The expected distance used in this work is defined using an integral and is normally computed by numerical integration such as the Monte Carlo methods. This kind of methods requires a sufficiently large number of samples (e.g., 10^5) to ensure accuracy. Hence, it will lead to very high time cost if we process queries by comparing distances of all objects one by one in real time. This naïve solution is intolerable for the vast majority of real-world applications which demand immediate responses. What is more, the world of today is being flooded with streams of large data and is in the age of big data. This calls for novel approaches that can handle a large dataset and support efficient query processing.

To find an efficient solution, we analyze properties of expected distance on Gaussian distribution mathematically and derive the lower bound and upper bound of this distance. Meanwhile, we employ the *filter-and-refine* paradigm to accelerate query processing. By filtering, we prune unpromising objects whose lower bound distances are larger than upper bound or expected distances of candidate objects without computing their actual distances. In refinement, we compute exact distances of candidate objects and finally return the top-k smallest ones. To further improve the performance, we utilize R-tree to index objects and their lower bound distances and upper bound distances. We propose three novel algorithms to support efficient query processing. The experimental result demonstrates that our proposed approaches can achieve great efficiency and are applicable to real-world applications.

We summarize our contributions as follows.

1. We formally define the problem of k-expected nearest neighbor search over objects represented by Gaussian distributions.
2. We analyze mathematically properties of the expected distance on Gaussian distribution and derive the lower bound and the upper bound of this distance.
3. We propose three novel approaches to improve the efficiency of query processing.
4. We demonstrate the efficiency of our approaches through a comprehensive experimental performance study.

The rest of this chapter is organized as follows. We formally define the problem in Section 3.2. Then in Section 3.3 we analyze expected distance on Gaussian distribution and derive its lower bound and upper bound. We propose three approaches in Section 3.4.

Experimental results and analyses are presented in Section 3.5. We review related work in Section 3.6. Finally, Section 3.7 concludes this chapter.

3.2 Problem Definition

In this work, we consider the problem of k-nearest neighbor search over Gaussian objects based on expected distance. We assume that all objects stored in the database are represented by Gaussian distributions with different parameters, i.e., their average vectors and covariance matrices are not the same. The query objects are fixed points in the same space.

A Gaussian distribution in the multi-dimensional space is represented by

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]$$

where μ denotes the average vector and $\boldsymbol{\Sigma}$ denotes the covariance matrix. $|\boldsymbol{\Sigma}|$ and $\boldsymbol{\Sigma}^{-1}$ represent the determinant matrix and reverse matrix of $\boldsymbol{\Sigma}$, respectively. $(\boldsymbol{x}-\boldsymbol{\mu})^T$ represents the transposition of $\boldsymbol{x}-\boldsymbol{\mu}$. For simplicity, we call objects represented by Gaussian distribution *Gaussian objects* afterwards.

Here, we focus on the *squared Euclidean distance* (SQED) since it is frequently used in many areas such as pattern mining and cluster analysis. The expected distance between a Gaussian object o and a query point q based on SQED (called *ESQED*) is

$$ESQED(o,q) = \int ||\boldsymbol{x}-\boldsymbol{q}||^2 \cdot p(\boldsymbol{x})d\boldsymbol{x}. \tag{3.1}$$

We formally define the problem as follows.

Definition 3.1. Given a database of Gaussian objects $\mathcal{D} = \{o_1,\ldots,o_n\}$, a query point q, a constant k, the problem of *k-Expected Nearest Neighbor Search over Gaussian objects* searches D for k Gaussian objects that are nearest to q based on Eq. (3.1).

The distance in Eq. (3.1) cannot be computed directly and is normally evaluated by numerical integration using the Monte Carlo methods. We employ the *importance sampling* [88] for efficiency. Specifically, we generate a sample x based on $p(x)$ and sum its SQED with q. The ESQED is the summed SQED divided by the number of samples generated. This computation requires a sufficiently large number of samples (e.g., 10^5) to ensure accuracy. Therefore, it will induce very high computation cost.

A straightforward solution is to do a sequential scan over the database and compute their distances with each query point. This method is obviously too time-consuming for real-world applications. In the preliminary experiments, we generate $100,000$ two-dimensional Gaussian objects and use $100,000$ samples for numerical integration. The average runtime of 100 queries is 1623.4 seconds, which is rather expensive and intolerable for the most of users.

In the following sections, we utilize the *filter-and-refine* paradigm to accelerate query processing. Specifically, we filter objects that are unlikely to become the result without computing their ESQEDs and refine candidate objects to obtain the final result. The filtering is to prune objects whose lower bound distances are larger than upper bound or expected distances of candidate objects. By refinement, we compute the ESQED of each candidate object and choose the top-k objects with the smallest ESQED as the result. Hence, it is important to derive the lower bound and the upper bound of ESQED (Section 3.3) and develop effective filtering algorithms (Section 3.4).

3.3 Lower bound and Upper bound

In [23], the following lemma is proved.

Lemma 3.2. *Given an object p with $f(x)$ as its probability density function and $\overline{p}$ as its centroid, for any point q we have*

$$\int ||\boldsymbol{p}-\boldsymbol{q}||^2 \cdot f(\boldsymbol{x})d\boldsymbol{x} = ||\overline{\boldsymbol{p}}-\boldsymbol{q}||^2 + \int ||\boldsymbol{x}-\overline{\boldsymbol{p}}||^2 \cdot f(\boldsymbol{x})d\boldsymbol{x}.$$

Based on this lemma, we can obtain

$$ESQED(o,q) = ||\boldsymbol{\mu} - \boldsymbol{q}||^2 + \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot p(\boldsymbol{x}) d\boldsymbol{x}.$$

In the subsequent discussion, we denote the second part as Δ^2, i.e.,

$$\Delta^2 = \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot p(\boldsymbol{x}) d\boldsymbol{x},$$

and derive its lower bound and upper bound. According to [19], the lower bound and upper bound of Gaussian distribution is as follows:

$$\begin{aligned} p^{\perp}(\boldsymbol{x}) &= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x}||^2}{2\lambda^{\perp}}\right] \\ p^{\top}(\boldsymbol{x}) &= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x}||^2}{2\lambda^{\top}}\right] \end{aligned}$$

where $\lambda^{\perp}$ and $\lambda^{\top}$ are the minimum and maximum eigenvalues of $\boldsymbol{\Sigma}$, respectively. Thus, the lower bound of Δ^2, denoted as Δ^2_{min}, is

$$\begin{aligned} \Delta^2 &\geq \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x} - \boldsymbol{\mu}||^2}{2\lambda^{\perp}}\right] d\boldsymbol{x} \\ &= \frac{d \cdot (\lambda^{\perp})^{1+d/2}}{|\boldsymbol{\Sigma}|^{1/2}} = \Delta^2_{min}. \end{aligned}$$

and its upper bound Δ^2_{max} is

$$\begin{aligned} \Delta^2 &\leq \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x} - \boldsymbol{\mu}||^2}{2\lambda^{\top}}\right] d\boldsymbol{x} \\ &= \frac{d \cdot (\lambda^{\top})^{1+d/2}}{|\boldsymbol{\Sigma}|^{1/2}} = \Delta^2_{max} \end{aligned}$$

For the proof, please refer to Appendix B.1.

Correspondingly, we obtain the lower bound (LB) and upper bound (UB) of ESQED:

$$LB(o,q) = ||\boldsymbol{\mu} - \boldsymbol{q}||^2 + \Delta^2_{min} \leq ESQED(o,q) \leq ||\boldsymbol{\mu} - \boldsymbol{q}||^2 + \Delta^2_{max} = UB(o,q). \quad (3.2)$$

3.4 Proposed Approaches

In this section, we propose three approaches for efficient query processing using the filter-and-refine paradigm described in Section 3.2. However, it still seems cumbersome and inefficient to compare the lower bound and upper bound distance of each Gaussian object. Our idea is to employ an R-tree to index all Gaussian objects so that most unpromising objects can be filtered by groups instead of individually. Moreover, we observe that Δ^2_{min} and Δ^2_{max} involve costly matrix decomposition but they do not depend on queries. Based on this observation, we consider precomputing Δ^2_{min} and Δ^2_{max} offline to speed up online query processing.

In the first approach, we first precompute Δ^2_{min} and Δ^2_{max} for each Gaussian object and then insert each d-dimensional Gaussian object into an R-tree as a $(d+1)$-dimensional region $[(\mu, \Delta^2_{min}), (\mu, \Delta^2_{max})]$. Here, (μ, Δ^2_{min}) is the lower left coordinate and (μ, Δ^2_{max}) is the upper right coordinate of the region. In this way, we can easily compute the lower bound and upper bound distance of each Gaussian object using Eq. (3.2) when queries come.

At the same time, we associate each Gaussian object with its covariance matrix so that we can compute its ESQED directly if it becomes a candidate. It should be noted that we only need to store the lower triangular part or the upper triangular part of each covariance matrix since it is symmetric. In other words, for a $d \times d$ covariance matrix, we store $d(d+1)/2$ out of its d^2 elements.

For an R-tree node N with a region of $[(\boldsymbol{\mu}_{min}, \Delta^2_{min}), (\boldsymbol{\mu}_{max}, \Delta^2_{max})]$, its lower bound and upper bound distance can be computed similarly using Eq. (3.2):

$$LB(N,q) = \|\boldsymbol{\mu} - \boldsymbol{q}\|^2_{min} + \Delta^2_{min} \leq ESQED(N,q) \leq \|\boldsymbol{\mu} - \boldsymbol{q}\|^2_{max} + \Delta^2_{max} = UB(N,q). \quad (3.3)$$

Below we use an example dataset of six two-dimensional Gaussian objects in Table 3.1 to describe our algorithms. $\mu_{i,1}$ (resp. $\mu_{i,2}$) denotes the average of each object oi in the first (resp. second) dimension. $\lambda_{i,1}$ (resp. $\lambda_{i,2}$) denotes the eigenvalue of the variance matrix of each object oi in the first (resp. second) dimension. $\Delta^2_{i,min}$ (resp. $\Delta^2_{i,max}$) denotes Δ^2_{min} (resp. Δ^2_{max}) of each object o_i.

o_i	$\mu_{i,1}$	$\mu_{i,2}$	$\lambda_{i,1}$	$\lambda_{i,2}$	$\Delta^2_{i,min}$	$\Delta^2_{i,max}$
o_1	0.5	4.0	0.5	1.0	0.71	2.83
o_2	2.9	5.8	1.0	1.0	2.00	2.00
o_3	4.0	6.0	1.5	0.5	0.58	5.20
o_4	6.7	2.0	1.0	1.5	1.63	1.84
o_5	8.5	3.0	0.5	0.5	1.00	1.00
o_6	9.0	1.5	0.5	1.0	0.71	2.83

TABLE 3.1: An example dataset

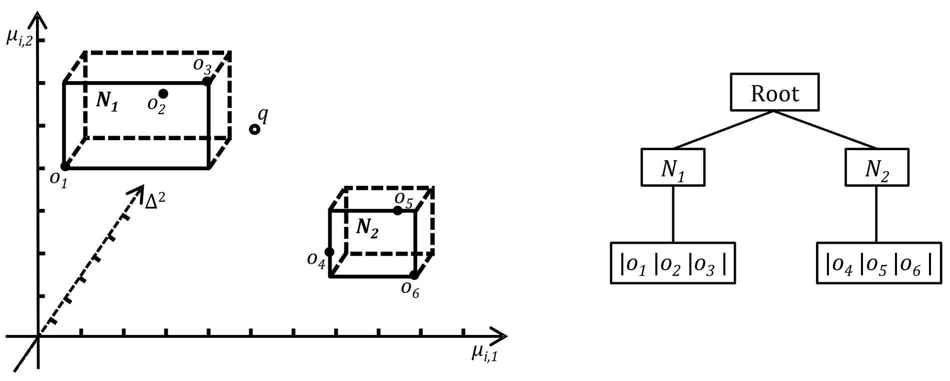


FIGURE 3.2: R-tree image and structure of the example dataset

In Fig. 3.2, we show the image and structure of the constructed R-tree on the example dataset and an example query. Each black solid point represents the average value in the first dimension $\mu_{i,1}$ and in the second dimension $\mu_{i,2}$ of each object o_i in the $(\mu_{i,1}, \mu_{i,2})$ plane. The example query is the black hollow point $q = (5.0, 5.0)$ with $k = 2$. We can easily compute $LB(N_1, q) = 0+1^2+0.58 = 1.58$, $LB(N_2, q) = 7.60$, $LB(o_1, q) = 21.96$, and $UB(o_3, q) = 7.20$. Notice that $UB(o_2, q) = LB(o_2, q) = 7.05$ because $\lambda_{2,1} = \lambda_{2,2}$. It means $ESQED(o_2, q) = 7.05$. In other words, the ESQED of a Gaussian object can be computed without numerical integration if its matrix covariance has the same eigenvalues in all dimensions. The same is to o_5, $UB(o_5, q) = LB(o_5, q) = 17.25$, i.e., $ESQED(o_5, q) = 17.25$.

Algorithm 3 shows the algorithm of our first approach. Since it is based on precomputing, LB, and UB, we call it PLUB afterwards. Entries (R-tree nodes or Gaussian objects) are processed in the ascending order of their lower bound distances. We utilize a priority queue Q to maintain candidate entries. After expanding the root node of the R-tree, entries in Q are $\{(1.58, N_1), (7.60, N_2)\}$. Then the algorithm continues to expand N_1. Since

Algorithm 3 The PLUB algorithm

```
1: Input: R-tree, $q$, $k$    ▷ Initialize the priority queue $Q$, the candidate set $C$, and the result set $R$
2: $Q \leftarrow$ Root of R-tree, $C \leftarrow \emptyset$, $R \leftarrow \emptyset$;
3: while $Q$ is not empty do
4:     $N \leftarrow Q$.top(); $Q$.pop();
5:     if $N$ is a leaf node then
6:         for each Gaussian object $o_i$ in $N$ do
7:             if $C$ has less than $k$ candidates then
8:                 Insert $o_i$ with $LB(o_i, q)$ and $UB(o_i, q)$ into $C$;
9:             else    ▷ $C$ has $k$ or more than $k$ candidates
10:                if $LB(o_i, q)$ is smaller than the $k$-th smallest UB in $C$ then
11:                    Insert $o_i$ with $LB(o_i, q)$ and $UB(o_i, q)$ into $C$;
12:    else    ▷ $N$ is an index node
13:        if $C$ has less than $k$ candidates then
14:            for each child entry $N_i$ in $N$ do
15:                Insert $N_i$ with $LB(o_i, q)$ into $Q$;
16:        else    ▷ $C$ has $k$ or more than $k$ candidates
17:            for each child entry $N_i$ in $N$ do
18:                if $LB(N_i, q)$ is smaller than the $k$-th smallest UB in $C$ then
19:                    Insert $N_i$ with $LB(o_i, q)$ into $Q$;
20:    if $C$ has $k$ or more than $k$ candidates and $LB(Q.top(), q)$ is not smaller than the $k$-th smallest UB in $C$ then
21:        break;
22: Compute the ESQEDs of the top-$k$ candidates with the smallest LB in $C$ and insert them with their ESQEDs into $R$
23: for each of the rest candidates $o_i$ in $C$ do
24:     if $LB(o_i, q)$ is smaller than the largest ESQED in $R$ then
25:         Compute $o_i$'s ESQED and replace it with $o_i$ and $o_i$'s ESQED;
26: return $R$;
```

N_1 is a leaf node and the candidate set C is empty, we insert o_1, o_2, and o_3 and their LBs and UBs into C. We obtain $C = \{(2.58, 7.20, o_3), (7.05, 7.05, o_2), (21.96, 24.08, o_1)\}$ and $Q = \{(7.60, N_2)\}$. The next top entry of Q, N_2, has an LB that is larger than the second largest UB in C, i.e., $7.60 > 7.20$, which is the second smallest upper bound distance in C. Thus, we finish processing Q and continue to update the result set R using C.

We first obtain the ESQED of o_3 by numerical integration since it LB is the smallest. For o_2, we do not have to compute its ESQED by numerical integration because we can directly obtain $ESQED(o_2, q) = 7.05$ as described above. By this time, $R = \{(3.99, o_3), (7.05, o_2)\}$. Since the LB of the last object o_1 in C, 21.96, is larger than 7.05, we return R and terminate the algorithm.

The second approach, called PLB, only utilizes the lower bound. After precomputing Δ^2_{min} for each Gaussian object, we insert each d-dimensional Gaussian object into an R-tree as a $(d + 1)$-dimensional point (μ, Δ^2_{min}). We can easily compute the lower bound distance of each Gaussian object using Eq. (3.2) when queries come. We also associate each Gaussian object with its covariance matrix so that we can compute its ESQED directly if it becomes a candidate. The lower bound distance of an R-tree node can be computed in the same way as PLUB using Eq. (3.3).

In the PLB algorithm, instead of inserting each candidate Gaussian object o_i with $LB(o_i, q)$ and $UB(o_i, q)$ into C (Line 8), we immediately compute its ESQED and insert o_i with $ESQED(o_i, q)$ into C. Moreover, we compare the LB of o_i, N_i, and $Q.top()$ with the k-th smallest ESQED rather than the k-th smallest UB in C (Line 10, 18, and 20). In Line 11, we replace the candidate having the k-th smallest ESQED of C with o_i if $ESQED(o_i, q)$ is smaller. Finally, since C has already had the top-k candidates with the smallest ESQEDs in this time we skip Line 22 – 25 and return C directly in Line 26.

Continuing with the example dataset in Table 3.1 and the example query of $q = (5.0, 5.0)$ with $k = 2$, in the PLB algorithm we expand the root node of the R-tree (the R-tree constructed by PLB is almost the same to that of PLUB in Fig. 3.2 and obtain $Q = \{(1.58, N_1), (7.60, N_2)\}$. When expanding N_1, we computes ESQEDs of o_1, o_2, and o_3 in turn and update $C = \{(3.99, o_3), (7.05, o_2)\}$ and $Q = \{(7.60, N_2)\}$. Then the algorithm will be terminated since the LB of the next top entry in Q is larger than the second largest ESQED in C, i.e., $7.60 > 7.05$. Finally, we return $C = \{(3.99, o_3), (7.05, o_2)\}$.

The idea of the PLB algorithm is that, using ESQED rather than UB would help to prune more entries since ESQED is always smaller and thus tighter than UB. At the same time, however, it may induce more ESQED computations since it immediately computes the ESQED of each candidate. For instance, we compute three times of ESQED using PLB while only one time using PLUB in the running example. We will compare the performance of PLUB and PLB in the experiment section.

The third approach does not rely on precomputing and directly utilizes the average of each Gaussian object, and is called AVG. Notice that by Eq. (3.2) we can obtain the *least lower bound* (LLB) of ESQED as follows:

$$LLB(o,q) = ||\boldsymbol{\mu} - \boldsymbol{q}||^2 < ||\boldsymbol{\mu} - \boldsymbol{q}||^2 + \Delta_{min}^2 \leq ESQED(o,q). \tag{3.4}$$

The least lower bound is looser than the lower bound but it allows us to filter unpromising objects using the simple average directly without caring about the complicated covariance matrix. we insert each d-dimensional Gaussian object into an R-tree as a d-dimensional point $\boldsymbol{\mu}$. We compute the least lower bound distance of each Gaussian object using Eq. (3.4) when queries come. We also associate each Gaussian object with its covariance matrix as we do previously. The least lower bound distance of an R-tree node is the minimum least lower bound distance of all its child nodes.

Algorithm 4 shows the AVG algorithm. It is very similar to the PLB algorithm except that it uses LLB as the search key instead of LB. We describe the algorithm by using the previous example in PLUB and PLB. The R-tree constructed by AVG is similar to the one in Fig. 3.2 other than that in this case the R-tree is two-dimensional. We first expand the root node of the R-tree and obtain $Q = \{(1.0, N_1), (6.89, N_2)\}$. Then we expand N_1 and computes ESQEDs of o_1, o_2, and o_3 in turn. Meanwhile, we update $C = \{(3.99, o_3), (7.05, o_2)\}$ and $Q = \{(6.89, N_2)\}$. Next, since the LLB of the next top entry in Q is smaller than the second largest ESQED in C, i.e., $6.89 < 7.05$, the algorithm cannot be terminated as PLUB and PLB do. In other words, we have to further expand N_2 and examine o_4, o_5, and o_6 through computing their ESQEDs. Finally, we return $C = \{(3.99, o_3), (7.05, o_2)\}$.

The AVG algorithm does not have to do any precomputing and the R-tree constructed by it has a lower dimension than of PLUB and PLB. As the performance of R-tree deteriorates with higher dimensions, we think this lower dimensional R-tree of AVG may compensate the cost of more ESQED computations caused by the looser least lower bound. We will verify this idea in the experiment section.

Algorithm 4 The AVG algorithm

```
1: Input: R-tree, $q$, $k$          ▷ Initialize the priority queue $Q$, the candidate set $C$
2: $Q \leftarrow$ Root of R-tree, $C \leftarrow \emptyset$;
3: while $Q$ is not empty do
4:     $N \leftarrow Q$.top(); $Q$.pop();
5:     if $N$ is a leaf node then
6:         for each Gaussian object $o_i$ in $N$ do
7:             if $C$ has less than $k$ candidates then
8:                 Insert $o_i$ with $ESQED(o_i, q)$ into $C$;
9:             else                                ▷ $C$ has $k$ or more than $k$ candidates
10:                if $LLB(o_i, q)$ is smaller than the $k$-th smallest ESQED in $C$ then
11:                    Replace it with $o_i$ and $ESQED(o_i, q)$;
12:    else                                         ▷ $N$ is an index node
13:        if $C$ has less than $k$ candidates then
14:            for each child entry $N_i$ in $N$ do
15:                Insert $N_i$ with $LLB(o_i, q)$ into $Q$;
16:        else                                     ▷ $C$ has $k$ or more than $k$ candidates
17:            for each child entry $N_i$ in $N$ do
18:                if $LLB(N_i, q)$ is smaller than the $k$-th smallest ESQED in $C$ then
19:                    Insert $N_i$ with $LLB(o_i, q)$ into $Q$;
20:    if $C$ has $k$ candidates and $LLB(Q.top(), q)$ is not smaller than the $k$-th smallest
       ESQED in $C$ then
21:        break;
22: return $C$;
```

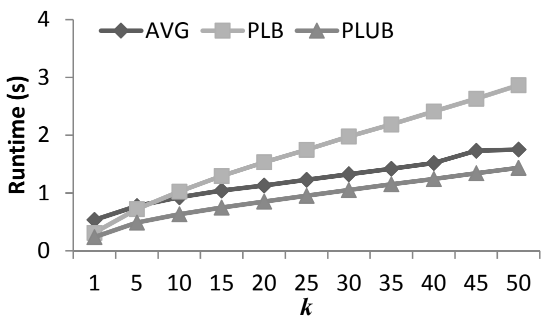
Parameter	Testing Range
k	1, 5, **10**, 15, 20, 25, 30, 35, 40, 45, 50
data size	10K, **100K**, 1M, 10M, 100M
dimension	**2**, 3, 4, 5, 6

Table 3.2: Parameters for testing

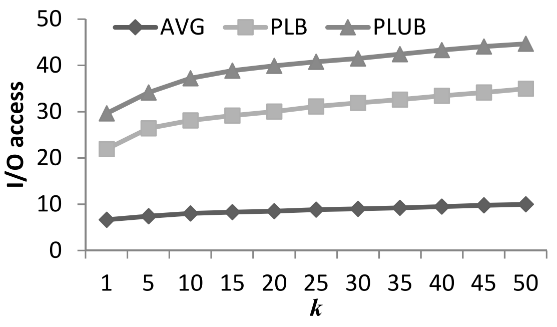
3.5 Experiments

3.5.1 Experimental Settings

We generate Gaussian distributions randomly for experiments. Each average value is generated from $(0, 1000)$, and each variance value is generated from $(0, 100)$. We check the effect of k, data size, and d , i.e., the number of result, the size of dataset, the dimensionality, on the performance of each approach. The parameters tested in our experiments and their default values (in bold) are summarized in Table 3.2.



(a) Runtime w.r.t *k*



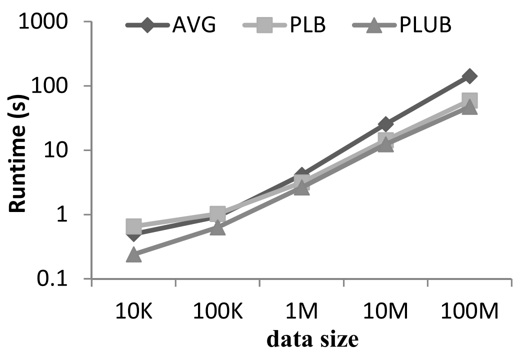
(b) I/O access w.r.t *k*

Figure 3.3: Effect of *k*

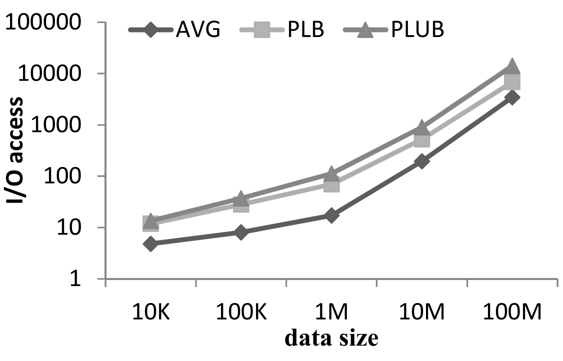
We compare the performance of our three proposed approaches: AVG, PLB, and PLUB. During preprocessing, we construct R-trees and store them in the secondary memory. In each experiment, we run 100 queries (generated randomly) and use the average runtime and I/O access for performance evaluation. All experiments are conducted using a workstation with Intel Xeon (R) CPU E3-1241 v3 (3.50GHz), RAM 16GB, running Ubuntu 12.10 LTS. Below we analyze the effect of each parameter on the performance of query processing and discuss the merits and demerits of the proposed approaches.

3.5.2 Effect of *k*

We show the average runtime and I/O access of the three approaches when varying *k* in Fig. 3.3(a) and Fig. 3.3(b). Their average runtime and I/O access all increases slowly with a larger *k*. This is because that we need to do more computations to retrieve more objects. As we can see from the two figures, PLUB runs the fastest, but its I/O access is also the highest. The reason is that most of objects accessed by PLUB will be pruned by upper bounds of candidate objects using their lower bounds and we only need to compute ESQEDs for a small part of them. Since the computation of ESQED consumes quite high time cost, this helps PLUB to achieve the best efficiency. In addition, when *k* is larger than 5, AVG performs better than PLB because the two-dimensional R-tree of AVG is more effective and retrieves less objects than the three-dimensional R-tree of PLB.



(a) Runtime w.r.t data size



(b) I/O access w.r.t data size

FIGURE 3.4: Effect of data size

3.5.3 Effect of Data Size

As shown in Fig. 3.4(a) and Fig. 3.4(b), when data size increases, the average runtime and I/O access of the three approaches all become larger. PLUB remains the most efficient approach while AVG becomes the slowest one when data size exceeds 100K though it has the least I/O access. This demonstrates that our derived lower bound and upper bound are very effective in improving the efficiency of query processing. The sublinear growth of runtime of our proposed approaches indicates that our approaches are applicable to large datasets.

3.5.4 Effect of Dimensionality

The dataset of the same size (100K) becomes sparser in a higher dimensional space. In other words, the object density decreases with increasing d. A smaller object density contributes to better performance of query processing since it becomes easier to prune unpromising objects. On the other hand, a higher d also means more complicated computation and thus slows down query processing. Fig. 3.5(a) and Fig. 3.5(b) show the effect of dimensionality on runtime and I/O access of the three approaches. When d is small, the impact of object density overrides that of computation, which results in decreasing runtime. When d is larger than 3, the impact of computation dominates the overall query processing and leads to increasing runtime. However, even when $d = 6$, the runtime is still less than 3 seconds (1 second for PLUB). Hence, our approaches can be applied to

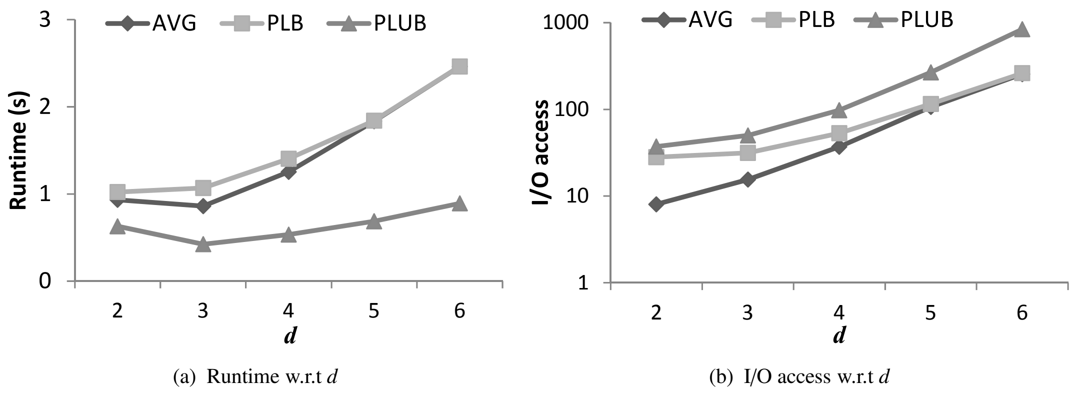


FIGURE 3.5: Effect of *d*

higher dimensions. This also indicates that PLUB is still best choice among the three approaches in terms of efficiency.

3.5.5 Discussion

Based on the above analyses, we conclude that PLUB achieves the best performance in efficiency because it uses both lower bound and upper bound for filtering. AVG is a better choice than PLB when *k* is larger than 10. But when data size is larger than 100K, PLB becomes to be preferred to AVG. In general, all of our proposed approaches can answer queries in seconds while the naïve way of sequential scan may take hours or even days. The index construction time of the three approaches does not differ very much with the same data size. For example, when data size is 100K, AVG, PLB, and PLUB takes about 0.266, 0.340, and 0.345 seconds, respectively. We can also see that the time used for index construction is quite small. PLB and PLUB have the same size of R-tree and their R-trees are slightly larger than that of AVG.

3.6 Related Work

Nearest neighbor search is a fundamental problem in such diverse areas that this problem has been studied extensively and is now relatively well understood. A survey on this problem is given in [89]. As uncertainty is inherent and immerging in many applications, there are a number of studies working on nearest neighbor searching under uncertainty [22, 23, 90], in which uncertainties are represented by probability distributions.

In [22], Ljosa *et al.* propose an approximate scheme to index *expected distance* for nearest neighbor search under the L1 distance. Their result cannot be applied to our problem since we consider the exact case under the *squared Euclidean distance*. In [23], Agarwal *et al.* also consider the expected distance from a query point to an uncertain object and return *expected nearest neighbor* (ENN). They propose efficient algorithms for answering ENN queries under several distance functions including squared Euclidean distance. However, their discussions focus on theoretic computation of the expected Voronoi diagram as preprocessing to answer ENN queries.

On the other hand, [90] considers the aspect of probability and studies *probabilistic nearest neighbor* (PNN) queries where the qualification probability of an object being the nearest neighbor of a query point is larger than a threshold (0 or a specified probability threshold). There is also much work studying top-*k* probable nearest neighbor [91], superseding nearest neighbor [92], and ranking queries [93].

However, none of the above work pays particular attention to Gaussian distribution, which is a popular probability distribution in many fields of applications. In [26], Böhm *et al.* model the feature vector of an uncertain object using Gaussian distribution and find similar Gaussian distributions to a given query Gaussian distribution. The restriction of this work is that they assume all Gaussian distributions are probabilistically independent in each dimension, which makes it difficult to be applied to generic Gaussian distributions and limits its overall accuracy of the query result. Given a query object (a point or Gaussian distribution), the work in [20] retrieves data objects represented by Gaussian distributions that are with a certain range from the query with probabilities larger than a specified threshold. They are both different from the problem we study in this work.

Finally, it is worth mentioning that nearest neighbor search has numerous variations such as reverse nearest neighbor search [94], aggregate nearest neighbor search [95], continuous nearest neighbor search [96], etc.

3.7 Summary

In this work, we considered k-nearest neighbor search over objects represented by Gaussian distributions based on expected distance. To support efficient query processing, we analyzed mathematically the properties of expected distance on Gaussian distribution and propose three novel approaches: AVG, PLB, and PLUB. We demonstrated the efficiency of our proposed approaches through extensive experiments and a comprehensive performance study. Among the three approaches, PLUB achieves the best efficiency while AVG is better than PLB if k is large, and PLB is a better choice in the case of a large data size.

Chapter 4

Top-*k* Similarity Search over Gaussian Distributions Based on KL-Divergence

4.1 Introduction

Probabilistic modeling, which infers probability distributions from vast amount of data for real-world applications, is being practiced in a wide range of fields from statistics, machine learning and pattern recognition [15] to bioinformatics and medical informatics [97]. The research on managing probabilistic model-based data was pioneered by Deshpande *et al.* [66], and then received considerable attention from the database research community [8, 9, 11]. In this work, we study the problem of processing similarity search queries over probabilistic model-based data, specifically, over objects represented by Gaussian distributions. As shown in Fig. 4.1, given a database of Gaussian distributions G and a query Gaussian distribution q, our objective is to find top-k Gaussian distributions from G that are similar to q. In this work, we study query processing over Gaussian distributions in view of their property of being probability distributions rather than using them to represent uncertainties in Chapter 2 and Chapter 3.

Gaussian distribution, one of the most typical probability distributions, is widely used in statistics, pattern recognition, and machine learning [15, 16]. Research work on Gaussian

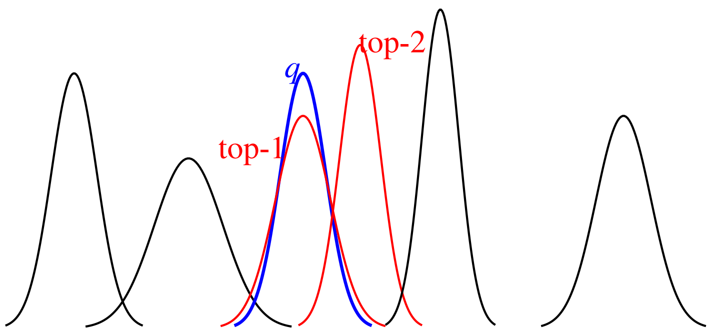


FIGURE 4.1: A one-dimensional query example ($k = 2$)

distributions has been conducted over a long period of time, including music classification [98] and search [99], and image retrieval [24, 100].

For this reason, we focus on similarity search over data modeled in Gaussian distributions, assuming that a large number of objects, represented by non-correlated Gaussian distributions are stored in the database. By *non-correlated Gaussian distribution*, we mean that all dimensions are independent with each other, i.e., the covariance matrix of each Gaussian distribution is diagonal. In this work, we focus on non-correlated Gaussian distributions since they are frequently used in machine learning and statistics. Hereafter, we use the term Gaussian distributions for non-correlated ones. We will report query processing methods for the general correlated case in the future work because they are very different. Given a query Gaussian distribution, our task is to retrieve from the database the Gaussian distributions that are similar to the query. The top-k Gaussian distributions with the highest similarity scores are returned as the answer to the query.

In [24], Böhm *et al.* considered similarity search on feature vectors such as structural features of 2-D contours [101], time series [102] and color histograms in image databases. They represented the uncertainty of each feature vector using a non-correlated multidimensional Gaussian distribution. As discussed in [15], compared to general Gaussian distributions, the number of parameters, the storage and computational requirements can be reduced substantially by using non-correlated Gaussian distributions. In consequence, the non-correlated Gaussian distribution is often preferred in practical applications [103].

Furthermore, Gaussian mixture models (GMMs), which are linear combinations of Gaussian distributions, are known for their ability to model arbitrarily shaped distributions.

For instance, GMMs are used to model driver behaviors for driver identification in [104]. We believe that our work paves the way for similarity search over GMMs, which will be beneficial for many real world applications such as finding drivers with similar driving behaviors.

To capture the similarity between a data Gaussian distribution and a query Gaussian distribution, we choose *Kullback-Leibler divergence* (KL-divergence) [27], a representative measure for quantifying the similarity between two probability distributions. KL-divergence is introduced in [105], and has then become the commonest divergence measures used in practice [106].

It is well-known that KL-divergence is a non-metric measure which violates the properties of a standard distance function in metric spaces such as the Euclidean space with the Euclidean distance. Specifically, it is asymmetric and does not satisfy the triangular inequality. Hence, existing index structures based on distance functions for metric spaces like M-tree [107] cannot be employed to solve this problem.

A naïve solution is to sequentially compute the KL-divergence with the query Gaussian distribution for each Gaussian distribution in the database, and select ones with top-*k* smallest KL-divergences. However, this method poses remarkable computing overhead and hence is not scalable to large datasets. In consequence, we employ the *filter-and-refine* paradigm to improve the efficiency of query processing. It first generates a set of promising candidates and filters unpromising ones without computing their similarities, and then candidate objects are refined to obtain the final results.

We propose two types of approaches utilizing the notions of *rank aggregation* [32] and *skyline queries* [33]. The first type presorts all objects in the database on their attributes and computes result objects by merging candidates from presorted lists. We modify the representative *threshold algorithm* (TA) [32] and propose two algorithms for efficient query processing. The second one transforms the problem to the computation of *dynamic skyline queries* [108]. We extend and modify the branch-and-bound skyline (BBS) algorithm [108], which is proposed to answer skyline queries, and develop a novel algorithm to solve this problem.

We note that although there have been several studies on searching in non-metric spaces [28–31], they are mainly developed for the generic class of non-metric similarity measures in discrete domains. None of them paid particular attention to the case where objects are modeled in Gaussian distributions and KL-divergence is chosen as the similarity measure. To the best of our knowledge, our work is the first study in similarity search based on KL-divergence over Gaussian distributions.

Our contributions are listed as follows.

1. We formalize the problem of top-k similarity search based on KL-divergence over Gaussian distributions, and analyze mathematically the properties of KL-divergence between two Gaussian distributions.
2. We propose two types of approaches to improve the efficiency of query processing using the notion of rank aggregation and skyline queries.
3. We demonstrate the efficiency and effectiveness of our approaches through a comprehensive experimental performance study.

The rest of this chapter is organized as follows. We formally define the problem in Section 4.2. Then we analyze KL-divergence of Gaussian distributions in Section 4.3 and propose two types of approaches in Section 4.4 and Section 4.5. Experimental results and analyses are presented in Section 4.6. We review related work in Section 4.7. Finally, Section 4.8 concludes this chapter.

4.2 Problem Definition

4.2.1 Gaussian Distribution

In the one-dimensional space, a Gaussian distribution is described by the average μ and the variance σ^2:

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right]. \tag{4.1}$$

In the d-dimensional space, it is represented by the average vector $\boldsymbol{\mu}$ and the covariance matrix $\boldsymbol{\Sigma}$ [15]:

$$p(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]. \tag{4.2}$$

$|\boldsymbol{\Sigma}|$ (resp. $\boldsymbol{\Sigma}^{-1}$) is the determinant (resp. inverse matrix) of $\boldsymbol{\Sigma}$. $(\cdot)^T$ means the transposition of $(\cdot)$. In this work, we assume that a large number of objects represented by Gaussian distributions are stored in the database. For simplicity, objects represented by Gaussian distributions are called *Gaussian objects*, and Gaussian objects in the database are called *data Gaussian objects* afterwards.

4.2.2 Similarity Measure: KL-divergence

Given two continuous probability distributions $p_1(x)$ and $p_2(x)$, the *Kullback-Leibler divergence* (KL-divergence) or *relative entropy* [27] between them is

$$\mathcal{D}_{\mathrm{KL}}(p_1 \| p_2) = \int_{-\infty}^{+\infty} p_1(x) \ln \frac{p_1(x)}{p_2(x)} dx. \tag{4.3}$$

In information theory, KL-divergence $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2)$ is interpreted as a measure of the inefficiency of assuming that the distribution is p_2 when the true distribution is p_1 [27]. In other words, it measures the information lost when p_2 is used to approximate p_1. The smaller the KL-divergence is, the more similar the two probability distributions are. Accordingly, the problem of *KL-divergence-based top-k similarity search over Gaussian distributions (KLD-Gauss)* is actually equivalent to finding top-k Gaussian objects having the smallest KL-divergences with the query Gaussian object.

KL-divergence satisfies the following properties of standard distance functions: 1) non-negativity: $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) \geq 0$; 2) identity: $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) = 0$ if and only if $p_1(x) = p_2(x)$. However, it is not symmetric, i.e., $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2) \neq \mathcal{D}_{\mathrm{KL}}(p_2\|p_1)$ in general. Moreover, it violates the notion of triangular inequality, namely, $\mathcal{D}_{\mathrm{KL}}(p_1\|p_2)+\mathcal{D}_{\mathrm{KL}}(p_2\|p_3) \geq \mathcal{D}_{\mathrm{KL}}(p_1\|p_3)$ does not necessarily hold. In other words, KL-divergence is a *non-metric* measure [31]. Hence, index structures designed for query processing in metric spaces such as M-tree [107] and iDistance [109] cannot be applied to accelerate similarity search based on KL-divergence.

As KL-divergence is asymmetric, given a data Gaussian object p and a query Gaussian object q, there are two options when using it as the similarity measure between them: $\mathcal{D}_{\text{KL}}(p||q)$ or $\mathcal{D}_{\text{KL}}(q||p)$. It is not easy to decide which one to use, and may vary according to different applications. Both of them are common in the literature. Thus, in this work, we study both types.

The naïve approach of solving the *KLD-Gauss* problem is to perform a sequential scan over all objects in the database and compute their KL-divergences for each query. This approach is obviously too time-consuming and will induce intolerable computation cost for many real-world applications, especially over large scale databases. To improve the efficiency of query processing, we adopt the well-known *filter-and-refine* paradigm. The rationale is to avoid unnecessary computations by developing effective filtering techniques.

In this work, we propose two types of approaches for filtering. They utilize the notions of *rank aggregation* [32] and *skyline queries* [33], respectively. Below we first present our analysis of KL-divergence of Gaussian distributions, and then introduce our approaches.

4.3 KL-divergence of Gaussian Distributions

Given two one-dimensional Gaussian distributions $p_1(x) = \mathcal{N}(\mu_1, \sigma_1)$ and $p_2(x) = \mathcal{N}(\mu_2, \sigma_2)$, their KL-divergence, denoted as $D^1_{\text{KL}}(p_1||p_2)$, is as follows [110]:

$$\mathcal{D}^1_{\text{KL}}(p_1||p_2) = \frac{1}{2}\left[\frac{(\mu_1-\mu_2)^2}{\sigma_2^2} + \frac{\sigma_1^2}{\sigma_2^2} - \ln\frac{\sigma_1^2}{\sigma_2^2} - 1\right]. \tag{4.4}$$

The two types of KL-divergence, $\mathcal{D}^1_{\text{KL}}(p||q)$ and $\mathcal{D}^1_{\text{KL}}(q||p)$, are referred to as $\mathcal{D}^1_{\text{KL1}}$ and $\mathcal{D}^1_{\text{KL2}}$, respectively.

In the d-dimensional space, given $p_1(x) = \mathcal{N}(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1)$ and $p_2(x) = \mathcal{N}(\boldsymbol{\mu}_2, \boldsymbol{\Sigma}_2)$, $D^d_{\text{KL}}(p_1||p_2)$ is defined by

$$\mathcal{D}^d_{\text{KL}}(p_1||p_2) = \frac{1}{2}\left[\sum_{i=1}^{d}\left(\frac{(\mu_{1,i}-\mu_{2,i})^2}{\sigma_{2,i}^2} + \frac{\sigma_{1,i}^2}{\sigma_{2,i}^2} - \ln\frac{\sigma_{1,i}^2}{\sigma_{2,i}^2}\right) - d\right], \tag{4.5}$$

where $\mu_{1,i}$ (resp. $\mu_{2,i}$) is the i-th ($1 \leq i \leq d$) element of p_1 (resp. p_2)'s average vector $\boldsymbol{\mu}_1$ (resp. $\boldsymbol{\mu}_2$). According to the independence assumption, $\boldsymbol{\Sigma}_1$ and $\boldsymbol{\Sigma}_2$ are diagonal matrices and their diagonal elements are denoted by $\sigma^2_{1,i}$ and $\sigma^2_{2,i}$, respectively. Obviously, $\mathcal{D}^d_{\mathrm{KL}} = \sum_{i=1}^{d} \mathcal{D}^i_{\mathrm{KL}}$, where $\mathcal{D}^i_{\mathrm{KL}}$ is the KL-divergence in the i-th dimension.

Similarly, the two types of $\mathcal{D}^d_{\mathrm{KL}}$, $\mathcal{D}^d_{\mathrm{KL}}(p\|q)$ and $\mathcal{D}^d_{\mathrm{KL}}(q\|p)$, are denoted by $\mathcal{D}^d_{\mathrm{KL1}}$ and $\mathcal{D}^d_{\mathrm{KL2}}$, respectively. Since they are sums of the one-dimensional case, their properties of monotonicity in each dimension are the same to that of $\mathcal{D}^1_{\mathrm{KL1}}$ and $\mathcal{D}^1_{\mathrm{KL2}}$, which will be discussed subsequently.

4.3.1 $\mathcal{D}^1_{\mathrm{KL}}(p\|q)$: $\mathcal{D}^1_{\mathrm{KL1}}$

As the smaller $\mathcal{D}^1_{\mathrm{KL1}}$ is, the more similar p is to q, we differentiate $\mathcal{D}^1_{\mathrm{KL1}}$ on p, specifically on μ_p and σ^2_p, and obtain the following equations:

$$\frac{\partial \mathcal{D}^1_{\mathrm{KL1}}}{\partial \mu_p} = \frac{\mu_p - \mu_q}{\sigma^2_q} \tag{4.6}$$

$$\frac{\partial \mathcal{D}^1_{\mathrm{KL1}}}{\partial \sigma^2_p} = \frac{\sigma^2_p - \sigma^2_q}{\sigma^2_p \sigma^2_q}. \tag{4.7}$$

By letting both Eq. (4.6) and Eq. (4.7) equal to 0, we derive the following property illustrated in Fig. 4.2. The arrows indicate decreasing directions of KL-divergence. We use $(\mu_p - \mu_q)^2$ as the horizontal axis to make the figure easy to understand.

Property 3. $\mathcal{D}^1_{\mathrm{KL1}}$ is a monotonically increasing function centered at (μ_q, σ^2_q) and divided by $\mu_p = \mu_q$, $\sigma^2_p = \sigma^2_q$.

1. As μ_p increases, $\mathcal{D}^1_{\mathrm{KL1}}$ increases monotonically when $\mu_p > \mu_q$, and decreases monotonically when $\mu_p < \mu_q$.

2. As σ^2_p increases, $\mathcal{D}^1_{\mathrm{KL1}}$ increases monotonically when $\sigma^2_p > \sigma^2_q$, and decreases monotonically when $\sigma^2_p < \sigma^2_q$.

3. $\mathcal{D}^1_{\mathrm{KL1}}$ is minimized at $\mu_p = \mu_q$, $\sigma^2_p = \sigma^2_q$, and its minimum is 0.

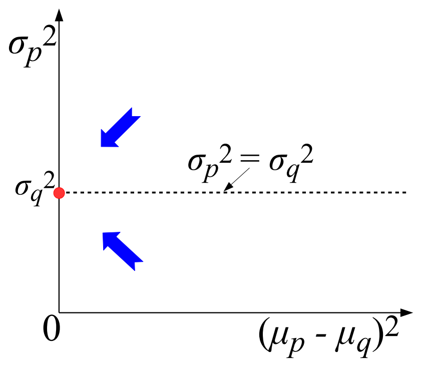


FIGURE 4.2: Property of $\mathcal{D}^1_{\mathrm{KL}1}$

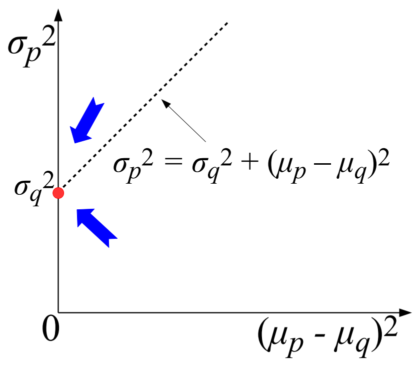


FIGURE 4.3: Property of $\mathcal{D}^1_{\mathrm{KL}2}$

Obviously, $\mathcal{D}^1_{\mathrm{KL}1}$ is divisionally monotonous. The closer a point is to the center (μ_q, σ_q^2) and the dividing lines $\mu_p = \mu_q$ and $\sigma_p^2 = \sigma_q^2$, the smaller $\mathcal{D}^1_{\mathrm{KL}1}$ is. In other words, smaller $|\mu_p - \mu_q|$ and $|\sigma_p^2 - \sigma_q^2|$ lead to smaller $\mathcal{D}^1_{\mathrm{KL}1}$.

4.3.2 $\mathcal{D}^1_{\mathrm{KL}}(q\|p)$: $\mathcal{D}^1_{\mathrm{KL}2}$

Similarly, we differentiate $\mathcal{D}^1_{\mathrm{KL}2}$ on μ_p and σ_p^2, and get the following equations:

$$\frac{\partial \mathcal{D}^1_{\mathrm{KL}2}}{\partial \mu_p} = \frac{\mu_p - \mu_q}{\sigma_p^2} \tag{4.8}$$

$$\frac{\partial \mathcal{D}^1_{\mathrm{KL}2}}{\partial \sigma_p^2} = \frac{\sigma_p^2 - \sigma_q^2 - (\mu_p - \mu_q)^2}{\sigma_p^4}. \tag{4.9}$$

In the same way, by letting both Eq. (4.8) and Eq. (4.9) equal to 0, we can obtain the following property illustrated in Fig. 4.3. It differs from $\mathcal{D}^1_{\mathrm{KL}1}$ in that its plane is divided by $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$ instead of $\sigma_p^2 = \sigma_q^2$.

Property 4. $\mathcal{D}^1_{\mathrm{KL}2}$ is a monotonically increasing function centered at (μ_q, σ_q^2) and divided by $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$.

1. As μ_p increases, $\mathcal{D}^1_{\mathrm{KL}2}$ increases monotonically when $\mu_p > \mu_q$, and decreases monotonically when $\mu_p < \mu_q$.

2. As σ_p^2 increases, $\mathcal{D}^1_{\mathrm{KL}2}$ increases monotonically when $\sigma_p^2 > \sigma_q^2 + (\mu_p - \mu_q)^2$, and decreases monotonically when $\sigma_p^2 < \sigma_q^2 + (\mu_p - \mu_q)^2$.

3. $\mathcal{D}^1_{\mathrm{KL2}}$ is minimized at $\mu_p = \mu_q$, $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$, and its minimum is 0.

Similarly, the closer a point is to the center (μ_q, σ_q^2) and the dividing lines $\mu_p = \mu_q$ and $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$, the smaller $\mathcal{D}^1_{\mathrm{KL2}}$ is. Hence, smaller $|\mu_p - \mu_q|$ and $|\sigma_p^2 - \sigma_q^2 - (\mu_p - \mu_q)^2|$ indicate smaller $\mathcal{D}^1_{\mathrm{KL2}}$.

4.4 TA-based Query Processing

The problem of sequential scan lies in that, it has to compute KL-divergences of all objects in the database, even though only k of them will be the answer. Based on this observation and monotonic properties of KL-divergence discussed in Section 4.3, we consider selecting a small number of promising candidate objects by presorting to avoid computing KL-divergences for unpromising objects. For example, in the one-dimensional case, we can sort the database on μ and σ^2 in advance, and only consider ones whose μ and σ^2 are close enough to that of q.

This is the basic idea of our first type of approaches, which utilize the notion of *rank aggregation* [32]. Generally speaking, we rank objects on each attribute and aggregate ones with high ranks to obtain the final top-k objects. Below we use the representative *threshold algorithm* (TA) [32] for description. To better solve the *KLD-Gauss* problem, we propose novel algorithms by modifying the TA algorithm. Below we first describe the TA algorithm and then introduce our proposed algorithms.

4.4.1 The TA Algorithm

TA assumes that a middleware system S aggregates answers to queries from various subsystems. Each subsystem S_i supports two modes of data access: *sorted access* and *random access*. In the *sorted access* mode, S obtains the grade of an object in the sorted list of S_i by proceeding through the list sequentially from the top. On the other hand, *random access* to S_i returns the corresponding grade of a given id.

Top images of subsystems

Rank	$s_{red}(x_i)$	$s_{round}(x_i)$
1	(0.96, x_1)	(0.95, x_2)
2	(0.91, x_2)	(0.92, x_3)
3	(0.85, x_4)	(0.85, x_5)
4	(0.81, x_3)	(0.83, x_4)
5	(0.72, x_5)	(0.76, x_1)

Top images in result

Rank	$s_Q(x_i)$, *attribute*
1	(x_2, 0.91), *red*
2	(x_4, 0.83), *round*
3	(x_3, 0.81), *red*

FIGURE 4.4: An example of the TA algorithm

For example, consider a database of images in Fig. 4.4 with two subsystems S_{color} and S_{shape}. S_{color} returns images based on their colors, and S_{shape} is based on shapes. Given a query of $Q : (color = \text{“}red\text{”}) \wedge (shape = \text{“}round\text{”})$, S merges images with high redness grades in S_{color} and high roundness grades in S_{shape} to obtain images satisfying the query.

Assume that Q requests top-3 images based on the score function $s_Q(x) = min\{s_{red}(x), s_{round}(x)\}$, where $s_{red}(x)$ (resp. $s_{round}(x)$) denotes the redness (resp. roundness) grade of image x. The left table lists top-5 images with their grades in each subsystem. Assume we retrieve one image by each sorted access. First, we obtain $\{s_{red}(x_1) = 0.96, s_{round}(x_2) = 0.95\}$. Other grades of x_1 and x_2, $s_{round}(x_1) = 0.76$ and $s_{red}(x_2) = 0.91$, can be obtained via random access. Thus, $s_Q(x_1) = 0.76$ and $s_Q(x_2) = 0.91$. The two images are both added into a result set $R = \{(x_2, 0.91), (x_1, 0.76)\}$. At the same time, the threshold τ is kept $\tau = min\{0.96, 0.95\} = 0.95$. This is the possible best score of all images unprocessed. Once scores of images in R are all no smaller than τ, the algorithm stops and returns R.

Next, $s_{red}(x_2) = 0.91$ and $s_{round}(x_3) = 0.92$ are retrieved. Since x_2's score has already been computed, we do random access only for x_3 and compute its score $s_Q(x_3) = 0.81$. Then x_3 is added into R and τ is updated to 0.91. In the next step, since x_4 has a better score than x_1, we update $R = \{(x_2, 0.91), (x_4, 0.83), (x_3, 0.81)\}$ and $\tau = 0.85$. Finally, as x_3 and x_4 have already been processed, we only need to update $\tau = 0.81$. At this point, τ is no better than any of images in R, i.e., no image unprocessed can have a better score than that in R. Therefore, the algorithm terminates and returns R as shown in the right table of Fig. 4.4. For ease of understanding, we associate each score with its corresponding attribute, i.e., red or round.

g_i	μ_i	σ_i^2
g_1	2	2.5
g_2	3.5	2.1
g_3	5	2.7
g_4	7	2.4
g_5	9	2.5
g_6	8	1.8
g_7	4	0.7
g_8	6	1.2
g_9	6.4	1.1
g_{10}	9	0.8
g_{11}	8.5	0.2
g_{12}	10.6	0.5

Order	S_μ	S_{σ^2}
1	$(2, g_1)$	$(0.2, g_{11})$
2	$(3.5, g_2)$	$(0.5, g_{12})$
3	$(4, g_7)$	$(0.7, g_7)$
4	$\mathbf{(5, g_3)}$	$\mathbf{(0.8, g_{10})}$
5	$(6, g_8)$	$\mathbf{(1.1, g_9)}$
6	$(6.4, g_9)$	$(1.2, g_8)$
7	$(7, g_4)$	$(1.8, g_6)$
8	$(8, g_6)$	$(2.1, g_2)$
9	$(8.5, g_{11})$	$(2.4, g_4)$
10	$(9, g_{10})$	$(2.5, g_1)$
11	$(9, g_5)$	$(2.5, g_5)$
12	$(10.6, g_{12})$	$(2.7, g_3)$

FIGURE 4.5: An example dataset and its sorted orders

4.4.2 The Proposed Algorithms

To utilize TA, we redefine *sorted access* and *random access* for the *KLD-Gauss* problem. Given an object id, *random access* returns the corresponding average vector and covariance matrix. We design two types of *sorted access*. The first one retrieves $\mu_{p,j}$ or $\sigma^2_{p,j}$ ($1 \leq j \leq d$) and object id in the ascending order of $|\mu_{p,j} - \mu_{q,j}|$ or $|\sigma^2_{p,j} - \sigma^2_{q,j}|$ (or $|\sigma^2_{p,j} - \sigma^2_{q,j} - (\mu_{p,j} - \mu_{q,j})^2|$, omit afterwards). The second one gives access to $\mathcal{D}^j_{\text{KL}}$ ($1 \leq j \leq d$) and object id in the ascending order of $\mathcal{D}^j_{\text{KL}}$. They are called *CompleteTA* (CTA) and *PartialTA* (PTA), respectively, and will be detailed subsequently.

4.4.2.1 The CTA Algorithm

For CTA, we presort the database on $\mu_{p,j}$ and $\sigma^2_{p,j}$ ($1 \leq j \leq d$), and get $2d$ sorted lists. For example, in Fig. 4.5 the left table shows a list of 12 one-dimensional objects, and the right table shows their sorted orders on μ_i and σ_i^2 (called S_μ and S_{σ^2}, respectively). In the multidimensional case, for each dimension j, we sort all objects on both $\mu_{i,j}$ and $\sigma^2_{i,j}$ and get $2d$ sorted lists: ($S_{\mu,1}$, $S_{\sigma^2,1}$), ..., ($S_{\mu,d}$, $S_{\sigma^2,d}$). By default, in each dimension j, the sorted access to each list of average returns an object p with its average $\mu_{p,j}$ in the ascending order of $|\mu_{p,j} - \mu_{q,j}|$, and the sorted access to each list of variance returns another object g with its variance $\sigma^2_{g,j}$ in the ascending order of $|\sigma^2_{g,j} - \sigma^2_{q,j}|$.

Algorithm 5 The straightforward CTA algorithm

```
1: Initialize the top-$k$ list $R$ and the query $q$ $(\mu_q, \sigma_q^2)$;
2: repeat
3:     for each dimension $j$ do
4:         $\texttt{SortedAccess}_{\text{avg}}() \rightarrow (\mu_{a,j}, g_a)$;
5:         $\texttt{SortedAccess}_{\text{var}}() \rightarrow \{(\overline{\sigma}_{v,j}^2, \overline{g}_v), (\underline{\sigma}_{v,j}^2, \underline{g}_v)\}$;
6:         if Any of $g_a, \overline{g}_v, \underline{g}_v$ has not been accessed then
7:             Do RandomAccess and calculate its KL-divergence;
8:             Update $R$;
9:     Let $\mu_j, \overline{\sigma}_j^2, \underline{\sigma}_j^2$ be the last values accessed by SortedAccess;
10:    $\mu \leftarrow \{\mu_j | 1 \le j \le d\}$; $\overline{\sigma}^2 \leftarrow \{\overline{\sigma}_j^2 | 1 \le j \le d\}$; $\underline{\sigma}^2 \leftarrow \{\underline{\sigma}_j^2 | 1 \le j \le d\}$;
11:    $\tau \leftarrow \min\{\texttt{CalcKLD}(\mu, \overline{\sigma}^2), \texttt{CalcKLD}(\mu, \underline{\sigma}^2)\}$;
12: until $|R| = k$ and KL-divergences in $R$ are all no greater than $\tau$;
13: return $R$;
```

Algorithm 5 shows the straightforward application of the TA algorithm using the redefined random access and sorted access. The algorithm runs by dimensions. In each dimension j, we retrieve an object g_a with its average $\mu_{a,j}$ by sorted access to the list of average. The average value of the retrieved object, $\mu_{a,j}$, is closest to that of the query, $\mu_{q,j}$, i.e., $|\mu_{a,j} - \mu_{q,j}|$ is the smallest. If there is a tie, i.e., there are two objects g_a and $g_{a'}$ satisfying $|\mu_{a,j} - \mu_{q,j}| = |\mu_{a',j} - \mu_{q,j}| = \Delta_\mu$, we can break it randomly since the algorithm relies on τ and the computation of τ depends on Δ_μ not $\mu_{a,j}$ or $\mu_{a',j}$.

On the other hand, because of the asymmetry of KL-divergence discussed in Section 4.2, each sorted access to the list of variance should return two objects in the two directions of $\overline{\sigma}_{v,j}^2 : \sigma_{v,j}^2 \ge \sigma_{q,j}^2$ and $\underline{\sigma}_{v,j}^2 : \sigma_{v,j}^2 < \sigma_{q,j}^2$ (one in each direction) instead of one object to ensure correctness. We explain it using the following example. Consider a query q with $\mu_q = 5$, $\sigma_q^2 = 1$ and $k = 3$. In the first step, while we retrieve $(5, g_3)$ from S_μ, from S_{σ^2} both $(1.1, g_9)$ and $(0.8, g_{10})$ need to be retrieved (in bold) to ensure correctness because we do not know which of them will have a smaller KL-divergence with respect to q. If we retrieve only $(1.1, g_9)$ and find $(5, 0.8)$ has a smaller KL-divergence than that of $(5, 1.1)$ with respect to q when computing τ, τ will be larger than it is supposed to be and this may lead to a wrong result. In other words, we start processing from the entries in bold and continue searching in both directions.

If an object is accessed for the first time by sorted access, we obtain its average vector and covariance matrix by random access and calculate its KL-divergence (Line 6 – 7).

Step	Retrieved objects	R	τ
1	g_3, (g_9, g_{10})	$(g_3, 0.35)$, $(g_9, 0.98)$, $(g_{10}, 8.01)$	0.002
2	g_7, (g_7, g_8)	$(g_3, 0.35)$, $(g_8, 0.508)$, $(g_7, 0.53)$	0.508
3	g_8, (g_6, g_{12})	$(g_3, 0.35)$, $(g_8, 0.508)$, $(g_7, 0.53)$	0.597

FIGURE 4.6: Query processing using the straightforward CTA

Step	Retrieved objects	R	τ
1	g_3, g_9	$(g_3, 0.35)$, $(g_9, 0.98)$	0.002
2	g_7, g_8	$(g_3, 0.35)$, $(g_8, 0.508)$, $(g_7, 0.53)$	0.508
3	g_8, g_{10}	$(g_3, 0.35)$, $(g_8, 0.508)$, $(g_7, 0.53)$	0.512
4	g_9, g_7	$(g_3, 0.35)$, $(g_8, 0.508)$, $(g_7, 0.53)$	1.01

FIGURE 4.7: Query processing using the improved CTA

For example, we do random access for g_3, g_9 and g_{10}, and compute their KL-divergences ($\mathcal{D}^1_{\text{KL1}}$ is used for computing KL-divergences in this example).

Then we update the top-k list R as follows. When $|R| < k$, the object with its KL-divergence is added to R directly. When $|R|$ achieves k, if its KL-divergence is better than any entry in R, we add it into R and delete the entry with the largest KL-divergence so that R only maintains the best k objects. Meanwhile, we compute the threshold τ using the last accessed average and variance values (Line 9 – 11). As τ is the best KL-divergence of all objects unseen, the algorithm terminates when KL-divergences in R are all no greater than τ.

We show the processing steps of the example query in Fig. 4.6. Continuing with the example, we update $R = \{(g_3, 0.35), (g_9, 0.98), (g_{10}, 8.01)\}$. At the same time, we compute KL-divergences of $(5, 1.1)$ and $(5, 0.8)$ with respect to the query $(5, 1)$, and update τ as the smaller one, which is 0.002.

In the second step, $(4, g_7)$ from S_μ, $(1.2, g_8)$ and $(0.7, g_7)$ from S_{σ^2}, are retrieved. Since KL-divergences of g_7 and g_8 are smaller than that of g_9 and g_{10}, we update R as $\{(g_3, 0.35), (g_8, 0.508), (g_7, 0.53)\}$ and $\tau = 0.508$. In the last step, we retrieve $(6, g_8)$ from S_μ, $(1.8, g_6)$ and $(0.5, g_{12})$ from S_{σ^2}. R stays the same, but τ is updated to 0.597. Finally, as all the objects in R have no greater KL-divergences than τ, we stop searching and return R.

In each step, as the straightforward algorithm retrieves two objects with respect to the variance in a brute-force way, it tends to do many unnecessary accesses. We avoid them by considering the priority of the object in each direction and access only one in each step. We derive the following lemma to guide the algorithm (see the proof in C.1).

Lemma 4.1. *1. Assume $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$ $(1 \le j \le d)$ and $\sigma^2_{p,m} = \sigma^2_{p',m}$ $(1 \le m \le d, m \ne j)$. If $\sigma^2_{p,j} > \sigma^2_{q,j}$, $\sigma^2_{p',j} < \sigma^2_{q,j}$, and $(\sigma^2_{p,j} - \sigma^2_{q,j}) \le (\sigma^2_{q,j} - \sigma^2_{p',j})$, then $\mathcal{D}_{\mathrm{KL}}(p\|q) < \mathcal{D}_{\mathrm{KL}}(p'\|q)$.*

2. Assume $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$ $(1 \le j \le d)$ and $\sigma^2_{p,m} = \sigma^2_{p',m}$ $(1 \le m \le d, m \ne j)$. If $\sigma^2_{p,j} > \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2$, $\sigma^2_{p',j} < \sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2$, and $(\sigma^2_{p,j} - \sigma^2_{q,j} - (\mu_{p,j} - \mu_{q,j})^2) \le (\sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2 - \sigma^2_{p',j})$, then $\mathcal{D}_{\mathrm{KL}}(q\|p) < \mathcal{D}_{\mathrm{KL}}(q\|p')$.

Based on Lemma 4.1, during the bidirectional search over the variance, when an object p with $\sigma^2_{p,j} > \sigma^2_{q,j}$ is obtained, we only need to consider another object p' with $\sigma^2_{p',j} < \sigma^2_{q,j}$, if $\sigma^2_{p',j}$ is nearer to $\sigma^2_{q,j}$ (or $\sigma^2_{q,j} + (\mu_{p',j} - \mu_{q,j})^2$) than that of p. For example, in the first step, in S_{σ^2} when comparing $(1.1, g_9)$ with $(0.8, g_{10})$, since $(1.1-1) \le (1-0.8)$, we only retrieve g_9 and delay the retrieval of g_{10}. In other cases, we compare their KL-divergences using the current average value μ_j obtained, i.e., KL-divergences of $(\mu_j, \sigma^2_{p,j})$ and $(\mu_j, \sigma^2_{p',j})$, and select the one with a smaller KL-divergence with respect to q.

We show the running steps of the improved CTA algorithm in Fig. 4.7. At first, we retrieve $(5, g_3)$ from S_μ. Meanwhile, we retrieve $(1.1, g_9)$ from S_{σ^2} based on Lemma 4.1 as explained above. Accordingly, we update R and τ. Then we continue to retrieve $(4, g_7)$ from S_μ. Since $(1.2 - 1) \le (1 - 0.8)$, we retrieve $(1.2, g_8)$ from S_{σ^2} based on Lemma 4.1 and update R and τ accordingly. In the third step, when comparing $(1.8, g_6)$ with $(0.8, g_{10})$ in S_{σ^2}, because they do not satisfy Lemma 4.1, we compute their KL-divergences using the current average value $\mu_8 = 6$. In other words, we compare KL-divergences of $(6, 1.8)$ and $(6, 0.8)$ with respect to $(5, 1)$. Then g_{10} is selected. As we can see, while the straightforward CTA algorithm accessed 7 objects, the improved one only retrieved 5 objects with one additional step.

4.4.2.2 The PTA Algorithm

For PTA, in each dimension we construct a two-dimensional R-tree of all data Gaussian objects. When a query q comes, by using the skyline-based approach described in Section 4.5, we can obtain the top objects with the smallest $\mathcal{D}^i_{\mathrm{KL}}$ with respect to q in each dimension i (detailed in Section 4.5.6). The PTA algorithm is similar to the CTA algorithm, except that we retrieve objects based on $\mathcal{D}^i_{\mathrm{KL}}$ instead of μ_i and σ^2_i with respect to q, and τ is calculated by $\tau = \sum_{i=1}^{d} \mathcal{D}^i_{\mathrm{KL}}$.

Next, we introduce another approach, which transforms the *KLD-Gauss* problem to *dynamic skyline queries* [108] by using the notion of *skyline queries* [33].

4.5 Skyline-based Query Processing

In this section, by utilizing the properties of KL-divergence and Gaussian distributions, we propose another approach for solving the *KLD-Gauss* problem. We extend and modify the BBS (Branch-and-Bound Skyline) algorithm [108], which is proposed to answer skyline queries. Below we first introduce the concept of skyline queries and the BBS algorithm, and then describe our extensions and the proposed algorithm.

4.5.1 Skyline Queries and Dynamic Skyline Queries

Given a dataset D, a skyline query [33] returns all the objects not *dominated* by others within D. p_i is said to be *dominated* by p_j, if p_j is better than p_i on at least one attribute, and better than or equals to p_i on other attributes. A common example is, given a list of hotels with prices and distances from a beach, to find ones having the lowest prices and the shortest distances. A hotel with \$200 and $1km$ from the beach is preferable to (*dominate*) the one of \$250 and $1.5km$.

A dynamic skyline query, a variation of a skyline query, returns all the objects that are not *dynamically dominated* with respect to a set of *dimension functions* $f_1, f_2, ..., f_m$ specified by the query [108]. Each function f_i takes as parameters a subset of the n attributes in the original n-dimensional space. The objective is to return the skyline in the new m-dimensional space defined by $f_1, f_2, ..., f_m$.

Continuing with the hotel example, assume for each hotel we store its x and y coordinates and price c (3-dimensional) information in the database. Dynamic skyline queries can be used to retrieve hotels that are closest to a specified location (x_u, y_u) and price c_u. In other words, closeness functions are defined by $f_1 = \sqrt{(x - x_u)^2 + (y - y_u)^2}$, and $f_2 = |c - c_u|$. Note that (x_u, y_u) and c_u normally vary with queries.

In [108], Papadias *et al.* proposed the BBS algorithm for processing skyline queries. They proved that BBS is I/O optimal; that is, it accesses the least number of disk pages among all algorithms based on R-trees. Hence, the following discussion concentrates on this technique.

4.5.2 The BBS Algorithm

Consider the task of computing the skyline of the example dataset in Fig. 4.5, i.e., finding objects with the smallest averages and variances. According to BBS, we construct an R-tree to index all objects. Each Gaussian object is inserted into the R-tree as a two-dimensional point (μ_i, σ_i^2). Its image and hierarchical structure are shown in Fig. 4.8 and Fig. 4.9. Each group of objects, i.e., an R-tree node, is represented by an MBR (Minimum Bounding Rectangle).

A priority queue Q is employed to maintain entries (R-tree nodes or Gaussian objects) to be accessed in the ascending order of *mindist*. The *mindist* of an entry, is the smallest cityblock (L_1) distance of its MBR to a reference point. For example, the *mindist* of N_1 to the origin O is calculated by summing up the length of OA and AB, where B is the lower-left point of N_1, and A is the projection of B on the μ_p-axis. The *mindist* of a Gaussian object to O, e.g., $g_1 = (2, 2.5)$, is simply its L_1 distance to O, i.e., $2 + 2.5 = 4.5$.

We use the example in Fig. 4.8 to illustrate the algorithm. After expanding the root node, entries in Q are $\{(N_1, 3.8), (N_2, 4.2)\}$. Then we expand N_1 and insert N_{11}, N_{12} into Q. Q becomes $\{(N_{11}, 4.1), (N_2, 4.2), (N_{12}, 8.8)\}$. When expanding N_{11}, since g_3 is dominated

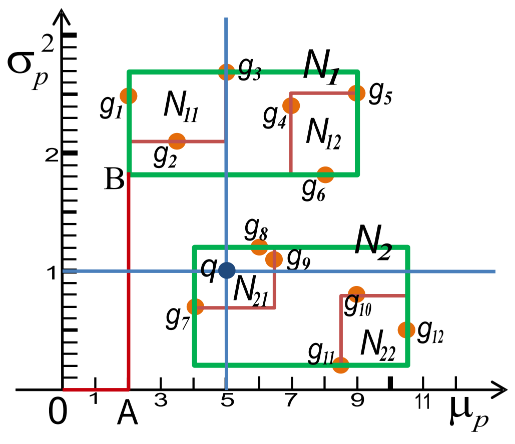


FIGURE 4.8: R-tree image

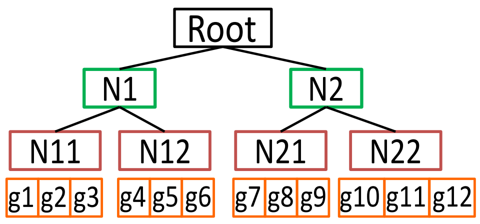


FIGURE 4.9: R-tree structure

by g_1 (and g_2), it is rejected. After expanding N_2, $Q = \{(g_1, 4.5), (N_{21}, 4.7), (g_2, 5.6), (N_{22}, 8.7), (N_{12}, 8.8)\}$.

Next g_1 is added into a candidate set S as the first skyline object. Since N_{21} is not dominated by g_1, it is expanded and g_7 is inserted into Q. Then $Q = \{(g_7, 4.7), (g_2, 5.6), (N_{22}, 8.7), (N_{12}, 8.8)\}$. g_7 and then g_2 are both added into S because g_7 is not dominated by g_1, and g_2 is not dominated by g_7 and g_1. Subsequently, expanding N_{22} leads to another candidate g_{11}. The last entry N_{12} will not be expanded as it is dominated by g_7. Finally, $S = \{g_1, g_7, g_2, g_{11}\}$ is returned as the skyline result.

BBS can be applied to compute dynamic skylines by expanding entries in Q according to *mindist* in the dynamic space [108]. In others words, we compute *mindist* with respect to the query object q instead of the origin O. Dynamic skylines can be computed in the same way except that the *mindist* of each entry in Q will be changed (each *mindist* is computed on-the-fly when the entry is considered for the first time). Assuming $f_1 = |\mu_p - \mu_q|$ and $f_2 = |\sigma_p^2 - \sigma_q^2|$, the dynamic skyline result of the query $q = (5, 1)$ in Fig. 4.8 is $\{g_3, g_8, g_9\}$.

4.5.3 Transformation and Extension

According to Section 4.3, the closer μ_p is to μ_q, σ_p^2 is to σ_q^2, the smaller KL-divergence is. This is analogous to a dynamic skyline query by assuming $f_1 = |\mu_p - \mu_q|$, $f_2 = |\sigma_p^2 - \sigma_q^2|$ (or

$f_2 = |\sigma_p^2 - \sigma_q^2 - (\mu_p - \mu_q)^2|$). Hence, we transform the *KLD-Gauss* problem to computing the dynamic skyline with respect to q.

In [108], BBS is applied directly to compute dynamic skylines. However, we note that since KL-divergence is *asymmetric* over σ_i^2, the two subspaces divided by $\sigma_p^2 = \sigma_q^2$ (or $\sigma_p^2 = \sigma_q^2 + (\mu_p - \mu_q)^2$) should be treated separately when we use BBS. In each dimension, we need to maintain two priority queues, and merge two result sets to obtain the final top-k objects. This is obviously inefficient and would incur extra computation cost.

Note that a straightforward method is to use iterative skyline computation according to the property that the top-1 object, based on any monotone ranking function, must be one of the skyline objects [111]. After deleting the top-1 object from our consideration, we recompute dynamic skyline objects. Then we select the top-2 object among them. Top-3, 4, . . ., k objects are computed in a similar way. This method would also lead to heavy cost.

The two concerns motivate us to propose a more efficient approach. We develop our ideas by extending the BBS algorithm. For ease of presentation, we describe our approach using $\mathcal{D}_{\mathrm{KL1}}^1$ and present necessary modifications in Section 4.5.5.

Because of the asymmetry of KL-divergence, we can only determine the dominance relationship between an object and each entry in the same subspace where the object locates, and cannot conclude whether it dominates any entry in the other subspace directly. For example, in Fig. 4.8 we can directly determine that g_8 dominates N_{12}. However, we do not know whether g_8 dominates g_7 or N_{22}. As a result, the dynamic skyline in each subspace has to be computed separately.

In order to reduce the cost caused by separate subspace computation, we consider enhancing the filtering power of each object from the subspace where it locates to the other subspace. If the dominance relationship between the two subspaces can be determined, we only need to maintain one priority queue and compute the two dynamic skylines together. To this end, for each point in one subspace, we find its counterpoint in the other subspace so that it can be used for filtering in that subspace.

This idea is confirmed by the properties of KL-divergence discussed in Section 4.3.1. Since KL-divergence shows monotonicity in both sides of $\sigma_p^2 = \sigma_q^2$ and is minimized at $\sigma_p^2 = \sigma_q^2$, for each point g in one subspace there must be one and only one corresponding point g' that satisfies $\mu_{g'} = \mu_g$ and $\mathcal{D}_{\mathrm{KL}}^1(g'||q) = \mathcal{D}_{\mathrm{KL}}^1(g||q)$. We call g' the *equal-KLD point* of g. The two equations produce $\sigma_{g'}^2 - \sigma_q^2 \ln \sigma_{g'}^2 = \sigma_g^2 - \sigma_q^2 \ln \sigma_g^2$. Because of its complexity, there is no analytical solution for $\sigma_{g'}^2$. We compute $\sigma_{g'}^2$ numerically by the *Newton method* and use its approximation (a slightly larger value to ensure correctness) instead. For example, the *equal-KLD point* of $g_8 = (6, 1.2)$ is $g_8' = (6, 0.82)$, which can be used to filter entries such as N_{22} additionally (i.e., g_8' dominates N_{22}).

We formally define the *equal-KLD point* as follows.

Definition 4.2 (*equal-KLD point*)**.** Given a Gaussian point g, g' is the *equal-KLD point* of g, if g' satisfies $\mathcal{D}_{\mathrm{KL}}^1(g'||q) = \mathcal{D}_{\mathrm{KL}}^1(g||q)$.

Accordingly, we extend the definition of *dynamically dominate* to *dynamically KLD-dominate* as follows.

Definition 4.3 (*dynamically KLD-dominate*)**.** Given a Gaussian point g and an entry e, g *dynamically KLD-dominates* e, if g's *equal-KLD point* g' *dynamically dominates* e.

For example, by computing g_8's *equal-KLD point* $g_8' = (6.01, 1.1)$, we can conclude that g_8 *dynamically KLD-dominates* $g_9 = (6.4, 1.1)$ since g_8' *dynamically dominates* g_9. As g is an *equal-KLD point* of itself, we can equally say g *dynamically KLD-dominates* e, if g *dynamically dominates* e. The relationship of g *dynamically KLD-dominates* e guarantees that the KL-divergence of g is smaller than that of any object within e.

To reduce the overhead of iterative skyline computation, we modify the way of dominance checking that the BBS algorithm does. We sort all candidates in the ascending order of KL-divergence. For each entry in the priority queue, we check the dominance relationship from the k-th candidate to the last one instead of checking all candidates as BBS does. If all entries are *dynamically KLD-dominated* by these “inferior” (from k to the last) candidates, the “superior” top-(k-1) and the first “inferior” candidates will be the final top-k result.

Algorithm 6 Skyline-based query processing algorithm

```
 1: procedure KLD_QUERY_SKY(R-tree, $q$, $k$)
 2:     $Q \leftarrow (0, Root)$;                      ▷ Initialize $Q$ with the Root of R-tree
 3:     $S \leftarrow \emptyset$;                      ▷ Initialize the candidate set
 4:     while $Q$ is not empty do
 5:         $e \leftarrow Q$.top(); $Q$.pop();
 6:         Check $e$ against $S$;
 7:         if $e$ is not dynamically KLD-dominated by $S$ then
 8:             if $e$ is a Gaussian object $g$ then
 9:                 Add $g$ and its KL-divergence into $S$;
10:             else                                   ▷ $e$ is an R-tree node $N$
11:                 for each child $N_i$ do
12:                     Check $N_i$ against $S$;
13:                     if $N_i$ is not dynamically KLD-dominated by $S$ then
14:                         $Q$.push($N_i$, $mindist(N_i)$);
15:     return top-$k$ of $S$;
```

4.5.4 The Proposed Algorithm: SKY

Based on the discussion above, we propose the SKY algorithm shown in Algorithm 6. An R-tree is employed to index averages and variances of all Gaussian objects in the database. Given the constructed R-tree, a query Gaussian object q, and a constant k, the algorithm returns the top-k Gaussian objects having the smallest KL-divergences with q.

The algorithm begins from the root node and continues until all entries in the priority queue Q are processed. When checking the top entry e of Q against the candidate set S (Line 6), if $|S| < k$, we add it to S if it is an Gaussian object, and expand it in the case of an R-tree node. Otherwise (i.e., $|S| \geq k$), we sort S in the ascending order of KL-divergence, and compare e against "inferior" candidates in S, i.e., from the k-th candidate until the last one. In this way, we can avoid the expensive iterative dynamic skyline computation. We reject e if it is *dynamically KLD-dominated* by S. If not, it is either added into S with its divergence (when e is a Gaussian object) or expanded (when e is an R-tree node). In the expansion of an R-tree node N, we check each of its child nodes N_i against S and insert N_i to Q if N_i is not *dynamically KLD-dominated* by S. Finally, the top-k candidates of S will become the result.

Another problem is that, there still will be quite a long list of entries in Q waiting for dominance checking. In order to prune non-promising entries, for each "inferior" candidate

p, we derive its *maximum mindist* (*mmdist*) using the *Lagrange multiplier*.

$$\begin{aligned} \underset{\mu_p, \sigma_p^2}{\text{maximize}} \quad & mmdist = |\mu_p - \mu_q| + |\sigma_p^2 - \sigma_q^2| \\ \text{subject to} \quad & \frac{1}{2}\left[\frac{(\mu_p - \mu_q)^2}{\sigma_q^2} + \frac{\sigma_p^2}{\sigma_q^2} - \ln\frac{\sigma_p^2}{\sigma_q^2} - 1\right] = C. \end{aligned}$$

To guide the algorithm based on *mmdist*, we further derive the following lemma (see the proof in C.2).

Lemma 4.4. *Any entry (Gaussian object or R-tree node) whose* mindist *is larger than* mmdist *of an object g will be* dynamically KLD-dominated *by g.*

According to Lemma 4.4, we only need to consider entries with *mindist* < *mmdist* so that the searching process can be finished early. We note that this filtering technique only works for $\mathcal{D}^1_{\text{KL1}}$. In other cases, all entries in Q have to be processed.

We use the example in Fig. 4.8 to illustrate our algorithm in the case of $\mathcal{D}^1_{\text{KL1}}$. Assume $k = 3$. After expanding the root node, $Q = \{(N_2, 0), (N_1, 0.8)\}$. Then we expand N_2 and $Q = \{(N_{21}, 0), (N_1, 0.8), (N_{22}, 3.7)\}$. Next, N_{21} is expanded and g_7, g_8, g_9 are inserted into Q. Entries in Q are $\{(N_1, 0.8), (g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (N_{22}, 3.7)\}$.

After expanding N_1, Q is $\{(N_{11}, 1.1), (g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (N_{12}, 2.8), (N_{22}, 3.7)\}$. Then N_{11}'s child objects g_1, g_2, g_3 are inserted into Q. Thus, $Q = \{(g_8, 1.2), (g_7, 1.3), (g_9, 1.5), (g_3, 1.7), (g_2, 2.6), (N_{12}, 2.8), (N_{22}, 3.7), (g_1, 4.5)\}$.

The top three entries g_8, g_7, g_9 with KL-divergences are added into S successively and $S = \{(g_8, 0.51), (g_7, 0.53), (g_9, 0.98)\}$. At the same time, we compute the *mmdist* of g_9, which is 4.03, and keep $\delta = 4.03$. Next, since g_3 is not *dynamically KLD-dominated* by g_9, it is inserted into S and $S = \{(g_3, 0.35), (g_8, 0.51), (g_7, 0.53), (g_9, 0.98)\}$. The *mmdist* of g_7 is $2.40 < 4.03$. Thus, δ is updated to 2.40. Since the *mindist* of g_2 is larger than δ, the algorithm stops and returns $\{(g_3, 0.35), (g_8, 0.51), (g_7, 0.53)\}$.

g_i	$\mu_{i,1}$	$\sigma^2_{i,1}$	$\mu_{i,2}$	$\sigma^2_{i,2}$
g_1	6.0	0.8	5.8	2.0
g_2	6.0	3.0	6.4	1.6
g_3	8.2	1.2	3.1	1.6
g_4	2.5	3.4	4.7	0.9

TABLE 4.1: A two-dimensional example dataset

4.5.5 Extending the Skyline-based Approach

The case of $\mathcal{D}^1_{\mathrm{KL2}}$ can be processed similarly except the filtering technique based on *mmdist* cannot be applied. In the multi-dimensional case, the algorithm is the same to the one shown in Algorithm 6. A $2d$-dimensional R-tree is constructed to index d-dimensional Gaussian objects in the database. The following definitions and computations will replace their counterparts in the one-dimensional case: (1) *mindist* $= \sum_{i=1}^{d}(|\mu_{p,i} - \mu_{q,i}| + |\sigma^2_{p,i} - \sigma^2_{p,i}|)$ (2) Compute *equal-KLD point* based on $\mathcal{D}^d_{\mathrm{KL1}}$ or $\mathcal{D}^d_{\mathrm{KL2}}$. (3) An object *g dynamically KLD-dominates* an entry *e*, if *g*'s *equal-KLD point g′ dynamically dominates e* in all dimensions.

4.5.6 Application to the PTA Algorithm

As discussed in Section 4.4.2.2, in each dimension PTA accesses the top objects with the smallest $\mathcal{D}^i_{\mathrm{KL}}$ using the skyline-based method SKY. Since PTA performs multiple sorted accesses in each dimension, we need to maintain all dominated entries instead of rejecting them as SKY does. We associate each "inferior" candidate with all entries dominated by it, and release them for further processing when this candidate becomes "superior" in the next sorted access.

We use the following example two-dimensional dataset shown in Table 4.1 to illustrate the PTA algorithm. We construct an R-tree in each dimension. Their R-tree images in the first dimension and the second dimension are shown in Fig. 4.10 and Fig. 4.11, respectively.

Consider the same query $q = (5.0, 2.0; 6.0, 1.5)$ with $k = 2$. Assume that we retrieve two objects in each dimension. At first, in the first dimension after expanding the root

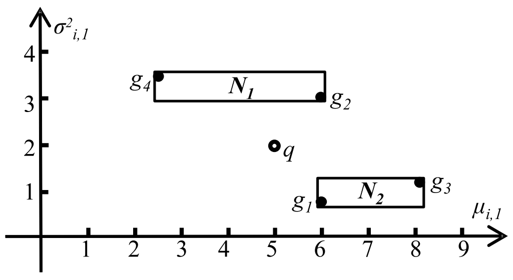


FIGURE 4.10: R-tree image (d = 1)

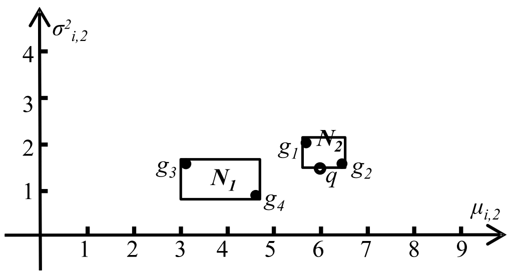


FIGURE 4.11: R-tree image (d = 2)

node in Fig. 4.10, we obtain $Q_1 = \{(N_1, 1.0), (N_2, 1.8)\}$ where Q_1 is a priority queue maintaining entries (R-tree nodes and Gaussian objects) to be processed in the ascending order of their *mindist*s from q. Then we expand N_1 and insert its two child Gaussian objects, g_2 and g_4, into $Q_1 = \{(N_2, 1.8), (g_2, 2.0), (g_4, 3.9)\}$. Next, we expand N_2 in the same way and obtain $Q_1 = \{(g_2, 2.0), (g_1, 2.2), (g_4, 3.9), (g_3, 4.0)\}$. The next entries, g_2 and g_1, are added into the candidate set S_1 successively with their KL-divergences in this dimension, $S_1 = \{(g_2, 0.297), (g_1, 0.408)\}$. At the same time, we compute the *mmdist* of g_1 and assign it to $\delta_1 = 3.904$. Since the *mindist* of the next entry g_4 is smaller than δ_1, i.e., $3.9 < 3.904$, we continue to check g_4. When checking it against S_1, we find that g_4 is *dynamically KLD-dominated* by g_1. Hence, we add it to the dominating list of g_1, and obtain $S_1 = \{(g_2, 0.297), (g_1, 0.408, \{(g_4, 3.9)\})\}$. At this time, we have $Q_1 = \{(g_3, 4.0)\}$. Since the *mindist* of the next entry g_3 is larger than δ_1, based on Lemma 4.4 we stop searching and return g_2 and g_1 with the second smallest $D^1_{KL} = 0.408$.

In the second dimension, we calculate the smallest D^2_{KL} in the same way as D^1_{KL} in the first dimension. After expanding the root node in Fig. 4.11, we obtain $Q_2 = \{(N_2, 0.1), (N_1, 1.3)\}$. Then we expand N_2 and obtain $Q_2 = \{(g_2, 0.5), (g_1, 0.7), (N_1, 1.3)\}$. Next, we add g_2 and g_1 into S_2 successively with their KL-divergences in this dimension, $S_2 = \{(g_1, 0.036), (g_2, 0.054)\}$. At the same time, we compute the *mmdist* of g_2 and assign it to $\delta_2 = 0.897$. Since the *mindist* of the next entry in $Q_2 = \{(N_1, 1.3)\}$ is larger than δ_2, again based on Lemma 4.4 we stop searching and return g_1 and g_2 with the second smallest $D^2_{KL} = 0.054$.

What follows is that the algorithm will compute the overall KL-divergences of g_2 and g_1 in the two dimensions (0.35 and 0.44), respectively, and update $\tau = D^1_{KL} + D^2_{KL} = 0.462$. Since the second smallest KL-divergence 0.44 is larger than τ, we first release $(g_4, 3.9)$

Parameter	Testing Range
k	1, **10**, 20, 30, 40, 50, 60, 70, 80, 90, 100
data size	1K, 10K, 100K, **1000K**, 10000K
dimension	1, **2**, 3, 4, 5
distribution	**independent**, correlated, anti-correlated

Table 4.2: Parameters for testing

from S_1 to Q_1 and delete the entries of g_1 and g_2 from S_1 and S_2, and then repeat the same process in the next round until the kth smallest KL-divergence is no larger than τ.

4.6 Experiments

4.6.1 Experimental Setup

We generate both data Gaussian distributions and query Gaussian distributions randomly under the same setting. Each average value is generated from (0, 1000), and each variance value is generated from (0, 100). The parameters tested in our experiments and their default values (in bold) are summarized in Table 4.2. To test the effect of data distribution, we also generate three datasets with independent, correlated, and anti-correlated distributions, respectively, using the standard skyline data generator in [33].

We compare our TA-based algorithms CTA, PTA and skyline-based algorithm SKY, with the sequential scan methods Scan and the extended BBS algorithm BBS. In each dimension i ($1 \le i \le d$), BBS processes objects in the two subspaces divided by $\sigma^2_{p,i} = \sigma^2_{q,i}$ (or $\sigma^2_{p,i} = \sigma^2_{q,i} + (\mu_{p,i} - \mu_{q,i})^2$) separately. In other word, it maintains 2^d priority queues and compute each top object by merging candidate objects from these priority queues.

During the preprocessing, we build a $2d$-dimensional R-tree for SKY and BBS. Each Gaussian distribution is inserted as a $2d$-dimensional point, consisting of the d-dimensional average vector and d-dimensional variance vector. For PTA, we construct d two-dimensional R-trees. Each two-dimensional R-tree maintains the average and variance value in the i-th dimension ($1 \le i \le d$), and provides *sorted access* to the ranked $\mathcal{D}^i_{\mathrm{KL}}$ and object id (see Section 4.4). *Random access* is supported by an Gaussian object list in the order of object id. All R-trees and lists are stored in the secondary memory.

In each experiment, we run 100 queries and use the average runtime for performance evaluation. We do not consider I/O access because the system tends to load indices of small size into main memory upon reading and the runtime is mainly spent on CPU. All experiments are conducted using a workstation with Intel Xeon(R) CPU E3-1241 v3 (3.50GHz), RAM 16GB, running Ubuntu 12.10 LTS.

4.6.2 Performance Evaluation

We first compare SKY with BBS. While SKY takes about 0.02 seconds using both the first type and the second type of KL-divergence (called KLD1 and KLD2 afterwards), BBS takes 2.45 seconds for KLD1, and 0.23 seconds for KLD2. This demonstrates the effectiveness of our extension and modification to the basic BBS algorithm. Because of the uneven property of KLD2 shown in Fig. 4.3, most of objects are assigned to the same priority queue. Consequently, BBS spends less time using KLD2 than that using KLD1. We leave BBS out of our consideration in the following experiments because of its clear inefficiency.

4.6.2.1 Effect of *k*

Intuitively, a larger k incurs more cost, since retrieving more objects potentially leads to more computations. The constantly increasing runtime of PTA, CTA and SKY shown in Fig. 4.12 follows this intuition. Scan is an exception of this intuition since it computes KL-divergence for all data Gaussian objects regardless of k. CTA and SKY outperform Scan over all k, and PTA performs better than Scan when $k < 100$ for KLD1 ($k < 60$ for KLD2). The difference is that PTA displays the most significant increase while the runtime of CTA rises more slowly, and SKY still keeps the low runtime as k increases. Even when k is 100, the runtime of SKY is only 12% of that of Scan.

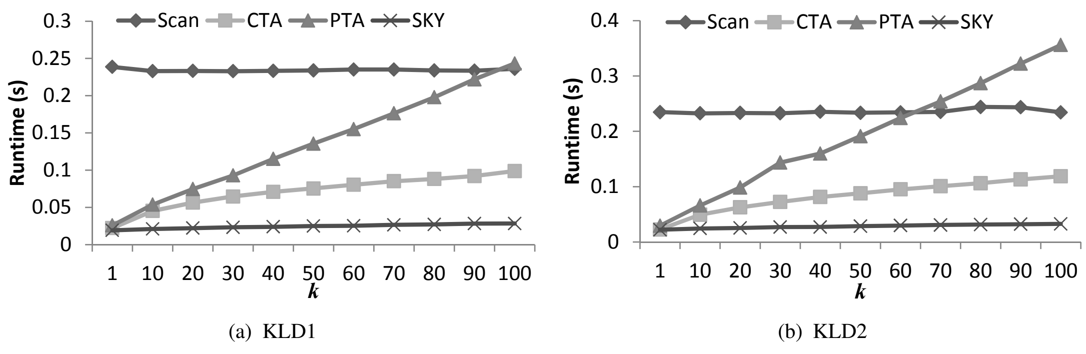


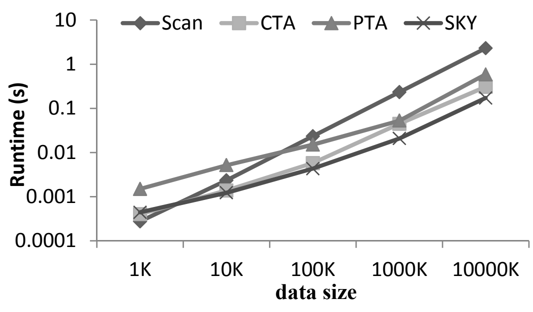
FIGURE 4.12: Effect of *k*

That's because the TA-based approaches, CTA and PTA, retrieve many objects iteratively based on their partial information in each dimension (average, variance or KL-divergence), and they need to retrieve much more objects with a larger *k*. On the other hand, SKY retrieves less objects based on their full information in all dimensions and thus costs less runtime. Especially, PTA needs to maintain all intermediate entries and compute the dynamic skyline to retrieve candidate objects in each dimension. Thus, it takes even more runtime than CTA, which retrieves candidates more easily by sorted access to presorted lists.

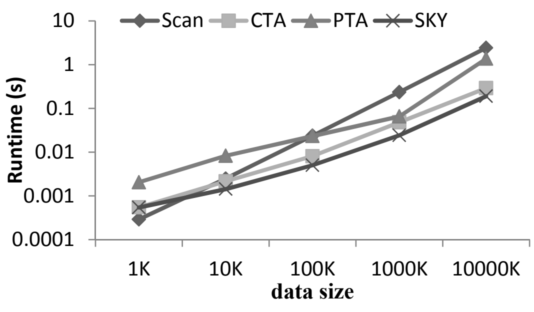
4.6.2.2 Effect of Data Size

Figure 4.13 shows the scalability of the four approaches. All of them consume more runtime with a larger data size. Scan exhibits the worst scalability to large scale datasets, while SKY demonstrates the best. The performance of PTA and CTA is between them with that CTA performs better.

When the data size increases, the density of objects in the space becomes larger. This means that there will be more objects with similar averages, variances or KL-divergences in one dimension but different in other dimensions. As a result, PTA and CTA suffer more from ineffective retrievals and thus perform worse than SKY, which has better scalability due to its more effective retrieval by R-tree. Especially, SKY and CTA outperform Scan when the data size is larger than 10K and are much better with the increasing data size.

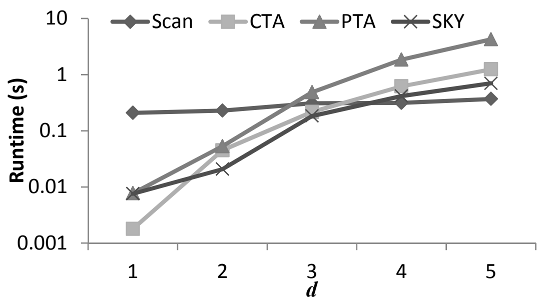


(a) KLD1

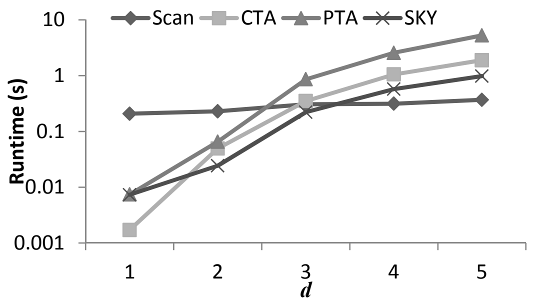


(b) KLD2

FIGURE 4.13: Effect of data size



(a) KLD1



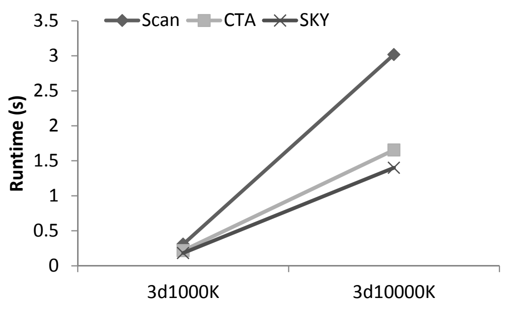
(b) KLD2

FIGURE 4.14: Effect of dimension

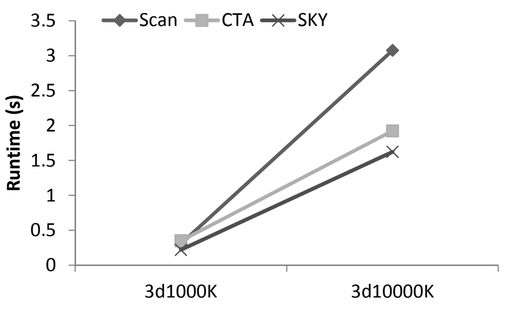
4.6.2.3 Effect of Dimensionality

As shown in Fig. 4.14, dimensionality affects the proposed approaches greatly, while it has little effect on Scan. Since PTA and SKY both utilize R-trees for indexing and the performance of R-tree degrades with the increasing dimension *d*, their runtime rises very fast. The runtime of PTA rises even faster since in each dimension it retrieves objects based on partial information and this results in much more ineffective retrievals, i.e., it retrieves much more unpromising objects. CTA spends more runtime with increasing *d* due to the similar reason as PTA that it does much more ineffective retrievals.

When *d* is larger than 3, they deteriorate significantly and are defeated by Scan. This indicates that our proposed approaches are more efficient in low dimensions less than 4. Moreover, their performance will be improved with a larger data size. For example, as shown in Fig. 4.15, when $d = 3$, the advantage of CTA and SKY over Scan is more obvious when the data size increases from 1000K to 10000K.

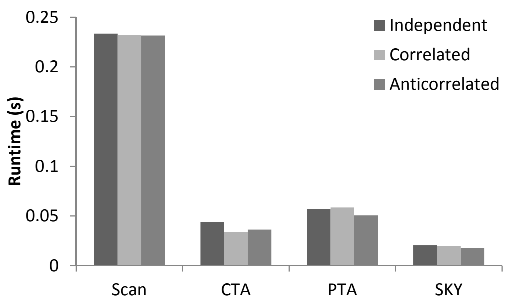


(a) KLD1

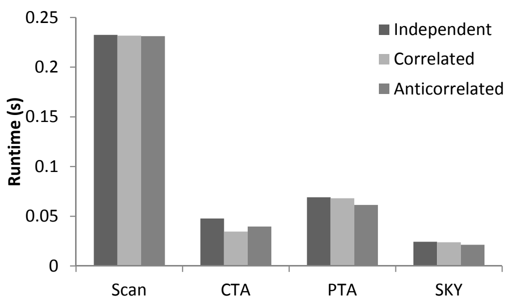


(b) KLD2

Figure 4.15: Improvement with increasing data size



(a) KLD1



(b) KLD2

Figure 4.16: Effect of data distribution

4.6.2.4 Effect of Data Distribution

In Fig. 4.16, we show the effect of data distribution on the four approaches. Generally, the runtime of all approaches does not vary greatly over the three different distributions. CTA shows the most apparent decrease over correlated and anti-correlated distributions since it depends directly on the average or variance of each object in each dimension. Correlations mean that objects are concentrated in the center and thus decrease the number of ineffective retrieval on average. It is clear that SKY has the best performance. This demonstrates the capability of our proposed approaches over different data distributions.

4.6.2.5 Index Construction

As preprocessing, we build R-trees and lists to support efficient query processing. In each experiment, we build a $2d$-dimensional R-tree for SKY and BBS, and d two-dimensional R-trees for PTA. When using the default dataset ($d = 2$, data size = 1000K), the index construction time is about 5 seconds for building a four-dimensional R-tree, and is about

7 seconds for building 2 two-dimensional R-trees. The lists include an object list for random access in CTA and PTA, and d sorted lists for sorted access in CTA and PTA. The time for building these two kinds of lists using the default dataset is about 1 second and 9 seconds, respectively. When the data size or d varies, the construction time varies proportionally. In view of the performance improvement shown in previous sections, the preprocessing cost of our proposed approaches is rather low.

4.7 Related Work

In this work, we proposed query processing approaches for top-k similarity search over Gaussian distributions based on KL-divergence, a *non-metric* similarity measure. Skopal *et al.* [31] surveyed the domains employing non-metric functions for similarity search, and methods for efficient and effective non-metric similarity search. To improve the searching efficiency, a class of approaches adopt the indexing strategy based on analytic properties. They analyze the properties of a specific similarity function, and develop special access methods for that function. Another advantage is their ability to support both exact and approximate search. Our work falls into this category and supports exact search.

Other approaches are based on statistical methods. For the efficiency, they perform costly preprocessing by suitably clustering or transforming the original database into another space such as the metric space based on the distance distribution statistics, so that existing metric indexing methods can be employed [28, 29, 31]. One drawback is that they cannot provide exact results. Furthermore, expensive preprocessing is often needed prior to the indexing itself. On the contrary, our proposed approaches have low preprocessing cost and support exact similarity search.

As a general class of similarity measures including KL-divergence, *Bregman divergence* has also led to a stream of research work to develop various algorithms. For example, [112] proposed clustering approaches with Bregman divergence. Zhang et al. [30] developed a prune-and-refine-based searching method for Bregman divergence, which is close to our work, but it works only for discrete probability distributions (described by

d-dimensional vectors) in finite domains. Moreover, they considered only the first type of asymmetric divergences.

In [24], an index structure called *Gauss-tree* was proposed for similarity search over multivariate Gaussian distributions. They also assumed non-correlated Gaussian distributions. Although their problem is very similar to ours, they defined a different similarity measure as follows. For a d-dimensional Gaussian distribution g, they used $N_{\mu_{g,j},\sigma_{g,j}}(x_j)$ to represent its probability density function in each dimension j, which is a one-dimensional Gaussian distribution with two parameters, the average $\mu_{g,j}$ and the variance $\sigma^2_{g,j}$. Given a database DB and a query Gaussian distribution q, the similarity of g in DB and q is defined as:

$$P(g|q) = \frac{p(q|g)}{\sum_{w \in DB} p(q|w)} \tag{4.10}$$

where

$$\begin{aligned} p(q|g) &= \prod_{j=1}^{d} \int_{-\infty}^{+\infty} N_{\mu_{q,j},\sigma_{q,j}}(x_j) \cdot N_{\mu_{g,j},\sigma_{g,j}}(x_j) dx \\ &= \prod_{j=1}^{d} N_{\mu_{q,j},\sigma_{q,j}+\sigma_{g,j}}(\mu_{g,j}) \end{aligned} \tag{4.11}$$

Here, $p(q|g)$ represents the probability density for observing q under the condition that we already observed g. The conditions that maximize Eq. (4.11) are $\mu_{g,j} = \mu_{q,j}$ and $\sigma_{g,j} \to 0$ not $\sigma^2_{g,j} = \sigma^2_{q,j}$. Hence, we think Eq. (4.10) is not a proper similarity measure for two Gaussian distributions.

4.8 Summary

In this work, assuming that large scale data is modeled by Gaussian distributions, we studied the problem of similarity search over non-correlated Gaussian distributions based on KL-divergence. We analyzed the mathematical properties of KL-divergence of Gaussian distributions. Based on the analysis, we proposed two types of approaches to efficiently and effectively process top-k similarity search over Gaussian distributions, which returns the k most similar ones to a given query Gaussian distribution. They utilize the notions of *rank aggregation* and *skyline queries*, respectively. We demonstrated the efficiency and effectiveness of our approaches through a comprehensive experimental performance study.

Chapter 5

Conclusions and Future Work

In this chapter, we conclude this thesis in Section 5.1 and present several interesting future directions in Section 5.2.

5.1 Conclusions

In this thesis, we studied uncertain data management due to the increasing uncertain data in a wide variety of fields in recent decades. Uncertainty can occur for different reasons and can be represented by a number of models. In particular, probabilistic model is utilized by many researchers to represent uncertainties using probability distributions. Among the three categories of probabilistic model, which are table-based, tuple-based, and attribute-based, we focused our attention on the third category, attribute-based, and further concentrate on the continuous case where the uncertain attribute is represented by a continuous probability density function, specifically, Gaussian distribution. In other words, we assumed that each uncertain object in the databases is described by a Gaussian distribution.

We considered three types of queries over probabilistic data with Gaussian distributions. The first one is *probabilistic range query*, which searches for objects within the given range with probabilities no less than a specified probability threshold. The query object can be either a certain point or an uncertain object represented by a Gaussian distribution. In the second work, we investigated *expected nearest neighbor search* based on the expected distance. The query object considered in this work is a certain point. The result objects are ones that have the k smallest expected distance to the query point. Finally, our third work studied *similarity search* over Gaussian distributions. The query object is also a Gaussian distribution. We use the KL-divergence as the similarity measure between two Gaussian distributions and return the top-k similar Gaussian distributions as the result.

Answering queries over uncertain databases poses a number of challenges since managing uncertainty usually means costly probability computations. Hence, we developed efficient solutions for these three problems and conducted comprehensive performance studies using both synthetic and real datasets. We summarized each of our proposed solutions as follows.

5.1.1 Probabilistic Range Queries

In this work, we studied probabilistic range queries over uncertain data represented by Gaussian distributions. We assumed that the locations of data objects are represented by Gaussian distributions and the location of the query object is either fixed or follows a multi-dimensional Gaussian distribution. Under this setting, we defined two types of probabilistic range queries with respect to the query object and called them *PRQ-P* and *PRQ-G* queries, respectively.

To reduce expensive probability computations, we proposed a set of filtering techniques to avoid unnecessary computations. We designed a number of filtering polices for both PRQ-P and PRQ-G queries based on approximated regions that are derived from our analysis of properties of Gaussian distribution. The proposed filtering techniques are very effective in reducing the number of objects that need to be verified by probability computation and thus can save computation cost.

Furthermore, we proposed a novel indexing method called G-tree to accelerate query processing. We developed the indexing method by extending the existing R-tree and improved it based on our analysis of Gaussian distribution. The indexing method organizes all the objects very effectively and enables filtering unpromising objects by groups rather than individually. Hence, the performance of query processing can be greatly enhanced.

The experimental results show that at least 99.5% of all the objects in the dataset can be pruned on average and over 58% of them are real result objects after evaluation by exact probability computation. This demonstrates the great pruning power of our proposed filtering techniques. Moreover, we equipped R-tree with our filtering techniques, called FR-tree, to evaluate the performance of our proposed indexing method G-tree. In the experiments, we compared them by varying the dataset size, the query range, the probability threshold, and the dimensionality. In all cases, G-tree outperforms FR-tree and shows great scalability and stability.

5.1.2 Expected Nearest Neighbor Search

In this work, we considered *k*-expected nearest neighbor search over objects represented by Gaussian distributions. This query finds the top-*k* objects that are nearest to a given query point based on their expected distances to the query point. In other words, the result objects are ones that have the *k* smallest expected distance to the query point. Since the naïve approach of performing sequential scan is computationally expensive for solving this problem, we proposed novel solutions to support efficient query processing.

We analyzed properties of expected distance on Gaussian distribution mathematically and derived the lower bound and upper bound of the distance. Based on our analysis, we proposed three novel approaches, AVG, PLB, and PLUB, to efficiently prune unpromising objects without computing their actual expected distances. We only compute actual expected distances for candidate objects and finally return the top-*k* smallest ones. To further improve the performance, we utilized R-tree to index objects and their lower bound distances and upper bound distances.

Both the lower bound and upper bound are described by the distance between the average point of a Gaussian distribution and the query point plus a non-negative minimum or maximum value that depends on the covariance matrix of the Gaussian distribution. The first one, AVG uses simply the distance between the average point and the query point as the lower bound to prune unpromising objects whose lower bound distances are greater than the actual expected distances of top-k candidates. This approach is rather simple, but it enables us to pay less cost on storing information about the covariance matrix in the R-tree. The second approach, PLB employs the more precise lower bound for filtering and needs to maintain more information about the covariance matrix. Finally, the third one, PLUB exploits both the lower bound and upper bound to prune non-candidates as much as possible. The non-candidates here are objects whose lower bound distances are greater than upper bound distances of candidate objects. Correspondingly, PLUB stores the most information about the covariance matrix. In other words, they are tradeoffs in different degrees between computation cost and storage cost.

In the experiments, we compared the performance of our proposed approaches AVG, PLB, and PLUB, by varying the three parameters, k, dataset size, and dimensionality. All the approaches show good scalability and stability over these parameters. We found that among the three approaches, PLUB achieves the best efficiency while AVG is better than PLB if k is large, and PLB is a better choice in the case of a large data size.

5.1.3 Similarity Search

In this work, we investigated similarity search on uncertain data modeled in non-correlated Gaussian distributions, where there are no correlations between dimensions and their covariance matrices are diagonal. The query object is also represented by a non-correlated Gaussian distribution. We employed KL-divergence to measure the similarity between two Gaussian distributions. This query returns the top-k Gaussian distributions that are similar to a given query Gaussian distribution based on KL-divergence. In other words, the result objects are Gaussian distributions that have the top-k smallest KL-divergence with the query Gaussian distribution.

We analyzed mathematically KL-divergence of Gaussian distributions and derived its divisionally monotonous properties. Based on the analysis, we proposed two types of approaches, TA-based and Skyline-based, to efficiently and effectively solve this problem. They utilize the notions of *rank aggregation* and *skyline queries*, respectively. The first type presorts all objects in the database on their attributes and computes result objects by merging candidates from each presorted list. We extended the existing TA algorithm and developed two novel algorithms, CTA and PTA. The second one transforms the problem to the computation of *dynamic skyline queries*. We extended and modified the BBS algorithm and developed a novel algorithm called SKY to solve this problem.

In our experimental study, we evaluated the effects of *k*, dataset size, dimensionality, and data distribution on our proposed three approaches, CTA, PTA, and SKY. The experimental results show that all of our approaches performs better than the naïve approach Scan. Among them, SKY demonstrates the best and the performance of PTA and CTA is between SKY and Scan, with that CTA performs slightly better. Moreover, we found that our proposed approaches show great stability over different data distributions.

5.2 Future Work

In this thesis, we considered the attribute-level uncertainty represented by Gaussian distribution and our proposed approaches are mainly based the spatial index R-tree. As one future direction, we can consider other uncertainty models such as other probability distributions and the discrete uncertainty model. Furthermore, we can consider the Gaussian Mixture model, which can be used to represent arbitrary complex probability distributions. Another future direction is to develop novel index structures that are able to be adapted to multiple uncertainty models. For instance, we can combine the power of R-tree with that of other indexing methods such as Grid-based index structure and Quadtree. In addition, we plan to design an integrated query framework that can support multiple types of queries at the same time.

Other interesting research directions include applying other types of queries such as skyline queries, join queries, reverse queries, and aggregate queries, over probabilistic data with Gaussian distributions and defining novel types of queries by observing and pondering over emerging real-world applications. Moreover, we can consider queries over probabilistic data with Gaussian distributions in distributed environments and road networks.

In the following sections, we present specific future work for each of our three work.

5.2.1 Probabilistic Range Queries

In the current implementation, the node split policy of G-tree follows that of R*-tree and the computation of the four penalty metrics (area, margin, overlap and centroid distance) used for splitting is based on that of TPR-tree. As the future work, the structure of G-tree can be optimized for more efficient query processing by taking advantage of its features. Furthermore, we plan to develop novel indexing methods based on other index structures such as the Grid-based index structure.

5.2.2 Expected Nearest Neighbor Search

Currently, we use directly R-tree to index Gaussian distributions and their lower bound and upper bound distances. In the future, we consider improving and extending R-tree for expected nearest neighbor search over Gaussian distributions. We also plan to consider other indexing methods such as the Grid-based index structure. Moreover, we consider adapting the indexing method proposed in our first work to support query processing in this work. In addition, we plan to extend our work to distance measures other than the expected distance.

5.2.3 Similarity Search

In the future, we plan to improve index structures to reduce preprocessing cost. In this work, we use directly R-tree to index Gaussian distributions. Therefore, in the future, we consider improving and extending R-tree for similarity search over Gaussian distributions. We also plan to consider other indexing methods such as the Grid-based index structure besides R-tree. Furthermore, we will study the similarity search problem in the general case of multi-dimensional Gaussian distributions and use similarity measures other than KL-divergence.

Appendix A

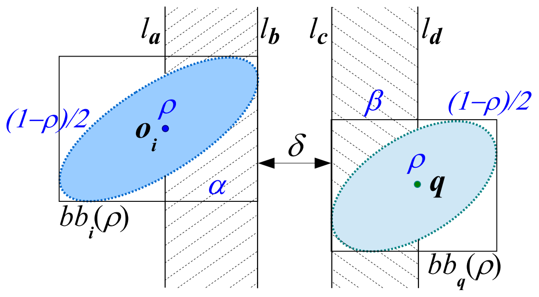
Appendix for Chapter 2

A.1 Proof of Lemma 2.3

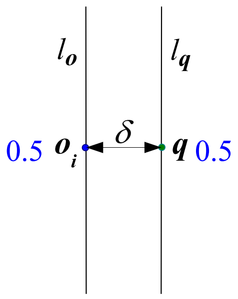
As shown in Fig. A.1(a), consider the case where the minimum distance between $bb_i(\rho)$ and $bb_q(\rho)$ is δ. The space outside the ρ-region of o_i (resp. q) is divided into two parts by the line l_a (resp. l_d) equally, both with an existence probability of $(1-\rho)/2$. Assuming the probability that o_i (resp. q) is located in the dashed area between l_a and l_b (resp. l_c and l_d) excluding the half part of ρ-region is α (resp. β), the probability that o_i lies within the left part of l_b and q lies within the left part of l_c is $((1-\rho)/2+\rho+\alpha)((1-\rho)/2-\beta)$. When o_i lies within the right part of l_b (q can be located both in the right and left part of l_c), the probability is $((1-\rho)/2-\alpha)$. Thus, the maximal probability of $\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta$ can be calculated by summing up the two probabilities, resulting in

$$\begin{aligned}
& Pr(\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta) \\
< \;& ((1-\rho)/2+\rho+\alpha)((1-\rho)/2-\beta) \\
+ \;& ((1-\rho)/2-\alpha) \\
= \;& (3-2\rho-\rho^2)/4-(\alpha+\beta)(1+\rho)/2-\alpha\beta \\
< \;& (3-2\rho-\rho^2)/4.
\end{aligned}$$

□



(a) $\theta < 0.75$



(b) $\theta \geq 0.75$

FIGURE A.1: Proof of maximal probability for PRQ-G

A.2 Proof of Lemma 2.4

Assume that the distance between the two mean locations is exactly δ as illustrated in Fig. A.1(b). o_i (resp. q) has a probability of 0.5 to be located in both left and right part of the line l_o (resp. l_q). $\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta$ happens in three cases: (1) Both o_i and q distribute in the left part of l_o and l_q. (2) Both o_i and q distribute in the right part of l_o and l_q. (3) o_i distribute in the right part of l_o and q distribute in the left part of l_q. Each case has a probability of $0.5 * 0.5$. Hence, the maximal probability of $\|\boldsymbol{x}_o - \boldsymbol{x}_q\| \leq \delta$ is $0.5 * 0.5 * 3 = 0.75$. $\square$

Appendix B

Appendix for Chapter 3

B.1 The Proof of Lower bound and upper bound of Δ^2

Since $\Delta^2 = \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot p(\boldsymbol{x})d\boldsymbol{x}$ and $p^{\perp}(\boldsymbol{x}) \leq p(\boldsymbol{x}) \leq p^{\top}(\boldsymbol{x})$, we have

$$\Delta^2_{min} = \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot p^{\perp}(\boldsymbol{x})d\boldsymbol{x} \leq \Delta^2 \leq \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot p^{\top}(\boldsymbol{x})d\boldsymbol{x} = \Delta^2_{max},$$

Here,

$$\begin{aligned} p^{\perp}(\boldsymbol{x}) &= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x}||^2}{2\lambda^{\perp}}\right] \\ p^{\top}(\boldsymbol{x}) &= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x}||^2}{2\lambda^{\top}}\right] \end{aligned}$$

where $\lambda^{\perp}$ and $\lambda^{\top}$ are the minimum and maximum eigenvalues of $\boldsymbol{\Sigma}$, respectively. We first derive Δ^2_{min} and then apply the same process to Δ^2_{max}.

Assume that in a d-dimensional space a vector $\boldsymbol{x}$ is represented by $(x_1, \ldots, x_d)$. By assuming $\boldsymbol{y} = \boldsymbol{x} - \mu$, we have

$$\begin{aligned} \Delta^2_{min} &= \int ||\boldsymbol{x} - \boldsymbol{\mu}||^2 \cdot \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{||\boldsymbol{x} - \boldsymbol{\mu}||^2}{2\lambda^{\perp}}\right] d\boldsymbol{x} \\ &= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \int ||\boldsymbol{y}||^2 \cdot \exp\left[-\frac{||\boldsymbol{y}||^2}{2\lambda^{\perp}}\right] d\boldsymbol{y}. \end{aligned} \tag{B.1}$$

According to [113], we transform Cartesian coordinates to spherical polar coordinates by

$$\begin{aligned}
y_1 &= r\cos\theta_1 \ldots \cos\theta_{d-2}\cos\theta_{d-1} = rc_1 \ldots c_{d-2}c_{d-1} \\
y_2 &= r\cos\theta_1 \ldots \cos\theta_{d-2}\sin\theta_{d-1} = rc_1 \ldots c_{d-2}s_{d-1} \\
&\vdots \\
y_k &= r\cos\theta_1 \ldots \cos\theta_{d-k}\sin\theta_{d-k+1} = rc_1 \ldots c_{d-k}s_{d-k+1} \\
&\vdots \\
y_1 &= r\sin\theta_1 = rs_1.
\end{aligned}$$

The Jacobian is $(-1)^d r^{d-1} c_1^{d-2} c_2^{d-3} \ldots c_{d-2}$. Then we have

$$\begin{aligned}
&\int \|\boldsymbol{y}\|^2 \cdot \exp\left[-\frac{\|\boldsymbol{y}\|^2}{2\lambda^\perp}\right] d\boldsymbol{y} \\
&= \int_{-\infty}^{+\infty} \ldots \int_{-\infty}^{+\infty} (\sum_{i=1}^{d} y_i^2) \cdot \exp\left[-\frac{1}{2\lambda^\perp} \sum_{i=1}^{d} y_i^2\right] dy_1 \ldots dy_d \\
&= \int_0^\infty \int_{-\pi/2}^{+\pi/2} \ldots \int_{-\pi/2}^{+\pi/2} \int_0^{2\pi} r^2 \cdot \exp\left[-\frac{r^2}{2\lambda^\perp}\right] \cdot \|(-1)^d r^{d-1} c_1^{d-2} c_2^{d-3} \ldots c_{d-2}\| dr d\theta_1 \ldots d\theta_{d-1} \\
&= \int_0^\infty r^{d+1} \cdot \exp\left[-\frac{r^2}{2\lambda^\perp}\right] dr \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \ldots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} \cdot \int_0^{2\pi} r^2 d\theta_{d-1}. \qquad \text{(B.2)}
\end{aligned}$$

According to [114],

$$\int_0^\infty x^n e^{-ax^b} dx = \frac{1}{b} a^{-\frac{n+1}{b}} \gamma(\frac{n+1}{b}).$$

By letting $n = d + 1$, $a = \frac{1}{2\lambda^b ot}$, and $b = 2$, we can obtain

$$\int_0^\infty r^{d+1} \cdot \exp\left[-\frac{r^2}{2\lambda^\perp}\right] dr = \frac{1}{2}(2\lambda^b ot)^{\frac{d+2}{2}} \gamma(\frac{d+2}{2}).$$

Then Eq. (B.2) becomes

$$\begin{aligned}
&\int \|\boldsymbol{y}\|^2 \cdot \exp\left[-\frac{\|\boldsymbol{y}\|^2}{2\lambda^\perp}\right] d\boldsymbol{y} \\
&= \frac{1}{2}(2\lambda^b ot)^{\frac{d+2}{2}} \gamma(\frac{d+2}{2}) \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \ldots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} \cdot \int_0^{2\pi} r^2 d\theta_{d-1} \\
&= (2\lambda^b ot)^{\frac{d+2}{2}} \gamma(\frac{d+2}{2}) \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \ldots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} \cdot \pi. \qquad \text{(B.3)}
\end{aligned}$$

Based on properties of Gamma function, the following statements hold.

1. If d is odd, then
$$\gamma(\frac{d+2}{2}) = \frac{d}{2} \cdot \frac{d-2}{2} \dots \frac{1}{2} \cdot \sqrt{\pi}. \tag{B.4}$$

2. If d is even, then
$$\gamma(\frac{d+2}{2}) = \frac{d}{2} \cdot \frac{d-2}{2} \dots \frac{2}{2} \cdot 1. \tag{B.5}$$

Also, when $n > 0$ we have

$$\begin{aligned}\int_{-\pi/2}^{+\pi/2} \cos^n \theta d\theta &= [\frac{\cos^{n-1}\theta \sin\theta}{n}]_{-\pi/2}^{\pi/2} + \frac{n-1}{n} \inf_{-\pi/2}^{\pi/2} \cos^{n-2}\theta d\theta \\ &= \frac{n-1}{n} \inf_{-\pi/2}^{\pi/2} \cos^{n-2}\theta d\theta.\end{aligned}$$

By deduction, the following statements hold.

1. If d is odd, then
$$\int_{-\pi/2}^{+\pi/2} \cos^n \theta d\theta = \frac{n-1}{n} \cdot \frac{n-3}{n-2} \dots \frac{4}{5} \cdot \frac{2}{3} \cdot 2.$$

2. If d is even, then
$$\int_{-\pi/2}^{+\pi/2} \cos^n \theta d\theta = \frac{n-1}{n} \cdot \frac{n-3}{n-2} \dots \frac{3}{4} \cdot \frac{1}{2} \cdot \pi.$$

Thus, we have

1. If d is odd, then
$$\begin{aligned}&\int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \dots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} \\ &= (\frac{d-3}{d-2} \cdot \frac{d-5}{d-4} \dots \frac{4}{5} \cdot \frac{2}{3} \cdot 2) \cdot (\frac{d-4}{d-3} \cdot \frac{d-6}{d-5} \dots \frac{3}{4} \cdot \frac{1}{2} \cdot \pi) \dots 2 \\ &= \frac{2\pi}{d-2} \dots \frac{2\pi}{3} \cdot 2.\end{aligned} \tag{B.6}$$

2. If d is even, then

$$\begin{aligned}
&\int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \dots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} \\
&= \left(\frac{d-3}{d-2} \cdot \frac{d-5}{d-4} \dots \frac{3}{4} \cdot \frac{1}{2} \cdot \pi\right) \cdot \left(\frac{d-4}{d-3} \cdot \frac{d-6}{d-5} \dots \frac{4}{5} \cdot \frac{2}{3} \cdot 2\right) \dots \frac{\pi}{2} \cdot 2 \\
&= \frac{2\pi}{d-2} \dots \frac{3\pi}{4} \cdot \pi. \qquad \text{(B.7)}
\end{aligned}$$

Based on Eq. (B.4), Eq. (B.5), Eq. (B.6), and Eq. (B.7), we have

1. If d is odd, then

$$\begin{aligned}
\gamma\left(\frac{d+2}{2}\right) \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \dots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} &= \left(\frac{d}{2} \cdot \frac{d-2}{2} \dots \frac{1}{2} \cdot \sqrt{\pi}\right) \cdot \left(\frac{2\pi}{d-2} \dots \frac{2\pi}{3} \cdot 2\right) \\
&= \frac{d}{2} \pi^{\frac{d}{2}-1}.
\end{aligned}$$

2. If d is even, then

$$\begin{aligned}
\gamma\left(\frac{d+2}{2}\right) \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \dots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} &= \left(\frac{d}{2} \cdot \frac{d-2}{2} \dots \frac{2}{2} \cdot 1\right) \cdot \left(\frac{2\pi}{d-2} \dots \frac{2\pi}{4} \cdot \pi\right) \\
&= \frac{d}{2} \pi^{\frac{d}{2}-1}.
\end{aligned}$$

In other words, the following equation holds no matter d is odd or even.

$$\gamma\left(\frac{d+2}{2}\right) \cdot \int_{-\pi/2}^{+\pi/2} c_1^{d-2} d\theta_1 \dots \int_{-\pi/2}^{+\pi/2} c_{d-2} d\theta_{d-2} = \frac{d}{2} \pi^{\frac{d}{2}-1}.$$

Then Eq. (B.3) becomes

$$\int ||\boldsymbol{y}||^2 \cdot \exp\left[-\frac{||\boldsymbol{y}||^2}{2\lambda^{\perp}}\right] d\boldsymbol{y} = (2\lambda^b ot)^{\frac{d+2}{2}} \cdot \frac{d}{2} \pi^{\frac{d}{2}-1} = d \cdot (\lambda^b ot)^{\frac{d}{2}+1} \cdot (2\pi)^{\frac{d}{2}}. \qquad \text{(B.8)}$$

Based on Eq. (B.1), and Eq. (B.8), we have

$$\Delta_{min}^2 = \frac{1}{(2\pi)^{d/2} |\boldsymbol{\Sigma}|^{1/2}} \cdot d \cdot (\lambda^b ot)^{\frac{d}{2}+1} \cdot (2\pi)^{\frac{d}{2}} = \frac{d \cdot (\lambda^b ot)^{1+d/2}}{|\boldsymbol{\Sigma}|^{1/2}}.$$

Similarly, we can prove

$$\Delta^2_{max} = \frac{d \cdot (\lambda^t op)^{1+d/2}}{|\mathbf{\Sigma}|^{1/2}}.$$

Hence, we have proved the lower bound and upper bound of Δ^2. □

Appendix C

Appendix for Chapter 4

C.1 Proof of Lemma 4.1

C.1.1 Case 1: $\mathcal{D}_{\mathrm{KL}}(p\|q)$

Assume $\sigma^2_{p,j} - \sigma^2_{q,j} = C_1$ and $\sigma^2_{q,j} - \sigma^2_{p',j} = C_2$, i.e. $\sigma^2_{p,j} = \sigma^2_{q,j} + C_1$ and $\sigma^2_{p',j} = \sigma^2_{q,j} - C_2$. Then $0 < C_1 \leq C_2 < \sigma^2_{q,j}$.

Let $\Delta_1 = \mathcal{D}_{\mathrm{KL}}(p\|q) - \mathcal{D}_{\mathrm{KL}}(p'\|q)$. Then

$$\begin{aligned}\Delta_1 &= \frac{1}{2}\ln\frac{\sigma^2_{q,j} - C_2}{\sigma^2_{q,j} + C_1} + \frac{C_1 + C_2}{2\sigma^2_{q,j}},\\ \frac{\partial \Delta_1}{\partial \sigma^2_{q,j}} &= \frac{(C_1 + C_2)[(C_2 - C_1)\sigma^2_{q,j} + C_1 C_2]}{2\sigma^4_{q,j}(\sigma^2_{q,j} - C_2)(\sigma^2_{q,j} + C_1)} > 0.\end{aligned}$$

Since $\Delta_1|_{\sigma^2_{q,j}\to\infty} = 0$, $\Delta_1 < 0$ holds for all $\sigma^2_{q,j}$, i.e., $\mathcal{D}_{\mathrm{KL}}(p\|q) < \mathcal{D}_{\mathrm{KL}}(p'\|q)$.

C.1.2 Case 2: $\mathcal{D}_{\mathrm{KL}}(q\|p)$

Since $|\mu_{p,j} - \mu_{q,j}| = |\mu_{p',j} - \mu_{q,j}|$, we use $|\mu_{p,j} - \mu_{q,j}|$ to represent both of them. Assume $\sigma^2_{p,j} - \sigma^2_{q,j} - (\mu_{p,j} - \mu_{q,j})^2 = C_1$ and $\sigma^2_{q,j} - \sigma^2_{p',j} - (\mu_{p,j} - \mu_{q,j})^2 = C_2$, i.e., $\sigma^2_{p,j} = \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2 + C_1$ and $\sigma^2_{p',j} = \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2 - C_2$. Then $0 < C_1 \leq C_2 < \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2$.

Let $\Delta_2 = \mathcal{D}_{\text{KL}}(q\|p) - \mathcal{D}_{\text{KL}}(q\|p')$ and $\alpha = \sigma^2_{q,j} + (\mu_{p,j} - \mu_{q,j})^2$. Then

$$\Delta_2 = \frac{1}{2}\ln\frac{\alpha + C_1}{\alpha - C_2} - \frac{\alpha(C_1 + C_2)}{2(\alpha + C_1)(\alpha - C_2)},$$
$$\frac{\partial\Delta_2}{\partial\sigma^2_{q,j}} = \frac{(C_1 + C_2)[\alpha(C_2 - C_1) + 2C_1C_2]}{2(\alpha - C_2)^2(\alpha + C_1)^2} > 0.$$

Since $\Delta_2|_{\sigma^2_{q,j}\to\infty} = 0$, $\Delta_2 < 0$ holds for all $\sigma^2_{q,j}$, i.e., $\mathcal{D}_{\text{KL}}(q\|p) < \mathcal{D}_{\text{KL}}(q\|p')$. □

C.2 Proof of Lemma 4.4

Let the *mmdist* of g be $mmdist_g$. Due to the definition of *mmdist*, for any *equal-KLD point* g' of g, $mmdist_g \geq |\mu_{g'} - \mu_q| + |\sigma^2_{g'} - \sigma^2_q|$ holds. Since $mindist_e$, the *mindist* of an entry e, satisfies $mindist_e = |\mu_e - \mu_q| + |\sigma^2_e - \sigma^2_q| > mmdist_g$, we have

$$|\mu_e - \mu_q| + |\sigma^2_e - \sigma^2_q| > |\mu_{g'} - \mu_q| + |\sigma^2_{g'} - \sigma^2_q|. \tag{C.1}$$

Given q and the KLD of g and q, and assuming

$$\mathcal{D}^1_{\text{KL}}(g\|q) = \frac{1}{2}\left[\frac{(\mu_g - \mu_q)^2}{\sigma^2_q} + \frac{\sigma^2_g}{\sigma^2_q} - \ln\frac{\sigma^2_g}{\sigma^2_q} - 1\right] = C,$$

when $\sigma^2_g = \sigma^2_q$, $(\mu_g - \mu_q)^2$ takes the maximum $2C\sigma^2_q$.

The reason is as follows. We can easily find that the function $f(x) = x - \ln x$ takes the minimum when $x = 1$. Therefore, $\frac{\sigma^2_g}{\sigma^2_q} - ln\frac{\sigma^2_g}{\sigma^2_q}$ takes the minimum when $\frac{\sigma^2_g}{\sigma^2_q} = 1$. In other words, when $\sigma^2_g = \sigma^2_q$, $(\mu_g - \mu_q)^2$ is the maximum. We prove the lemma in the following two cases.

1. $(\mu_e - \mu_q)^2 > 2C\sigma^2_q$:

   Consider g's *equal-KLD point* g' satisfying $|\mu_{g'} - \mu_q| = \sqrt{2C}\sigma_q$, and $\sigma^2_{g'} = \sigma^2_q$. Since $|\mu_e - \mu_q| > \sqrt{2C}\sigma_q = |\mu_{g'} - \mu_q|$, and $|\sigma^2_e - \sigma^2_q| \geq 0 = |\sigma^2_{g'} - \sigma^2_q|$, it is obvious that e is *dynamically dominated* by g', i.e., e is *dynamically KLD-dominated* by g.

2. Otherwise:

   Consider g's *equal-KLD point* g' satisfying $\mu_{g'} = \mu_e$. According to eq.(C.1), we have $|\sigma_e^2 - \sigma_q^2| > |\sigma_{g'}^2 - \sigma_q^2|$. In other words, $|\mu_e - \mu_q| = |\mu_{g'} - \mu_q|$, and $|\sigma_e^2 - \sigma_q^2| > |\sigma_{g'}^2 - \sigma_q^2|$. Therefore, e is *dynamically dominated* by g', i.e., e is *dynamically KLD-dominated* by g.

$\square$

Bibliography

[1] Charu C. Aggarwal and Philip S. Yu. A survey of uncertain data algorithms and applications. *IEEE TKDE*, 21(5):609–623, 2009.

[2] Nilesh N. Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52(7):86–94, 2009.

[3] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-driven data acquisition in sensor networks. In *VLDB*, pages 588–599, 2004.

[4] Charu C. Aggarwal and Philip S. Yu, editors. *Privacy-Preserving Data Mining - Models and Algorithms*. Springer, 2008.

[5] Vibhor Rastogi, Sungho Hong, and Dan Suciu. The boundary between privacy and utility in data publishing. In *VLDB*, pages 531–542, 2007.

[6] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[7] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *VLDB*, pages 965–976, 2006.

[8] Charu C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.

[9] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. *Probabilistic Databases*. Morgan & Claypool Publishers, 2011.

[10] Yijie Wang, Xiaoyong Li, Xiaoling Li, and Yuan Wang. A survey of queries over uncertain data. *Knowledge and Information Systems*, 37(3):485–530, 2013.

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

[12] Lyublena Antova, Christoph Koch, and Dan Olteanu. Query language support for incomplete information in the maybms system. In *VLDB*, pages 1422–1425, 2007.

[13] Sarvjeet Singh, Chris Mayfield, Sagar Mittal, Sunil Prabhakar, Susanne E. Hambrusch, and Rahul Shah. Orion 2.0: native support for uncertain data. In *ACM SIGMOD*, pages 1239–1242, 2008.

[14] Yufei Tao, Xiaokui Xiao, and Reynold Cheng. Range search on multidimensional uncertain data. *ACM TODS*, 32(3):15:1–15:54, 2007.

[15] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[16] Richard O. Duda, Peter E. Hart, and David G.Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

[17] Pankaj K. Agarwal, Siu-Wing Cheng, and Ke Yi. Range searching on uncertain data. *ACM Trans. Algorithms*, 8(4):43:1–43:17, 2012.

[18] Kostas Patroumpas, Marios Papamichalis, and Timos K. Sellis. Probabilistic range monitoring of streaming uncertain positions in geosocial networks. In *SSDBM*, pages 20–37, 2012.

[19] Yoshiharu Ishikawa, Yuichi Iijima, and Jeffrey Xu Yu. Spatial range querying for Gaussian-based imprecise query objects. In *ICDE*, pages 676–687, 2009.

[20] Tingting Dong, Chuan Xiao, and Yoshiharu Ishikawa. Probabilistic range querying over Gaussian objects. *The Institute of Electronics, Information and Communication Engineers Transactions*, 97-D(4):694–704, 2014.

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

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

[23] Pankaj K. Agarwal, Alon Efrat, Swaminathan Sankararaman, and Wuzhou Zhang. Nearest-neighbor searching under uncertainty. In *ACM PODS*, pages 225–236, 2012.

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

[25] Yufei Tao, Reynold Cheng, Xiaokui Xiao, Wang Kay Ngai, Ben Kao, and Sunil Prabhakar. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, pages 922–933, 2005.

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

[27] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley, 2006.

[28] Tomás Skopal. On fast non-metric similarity search by metric access methods. In *EDBT*, pages 718–736, 2006.

[29] Lei Chen and Xiang Lian. Efficient similarity search in nonmetric spaces with local constant embedding. *IEEE TKDE*, 20(3):321–336, 2008.

[30] Zhenjie Zhang, Beng Chin Ooi, Srinivasan Parthasarathy, and Anthony K. H. Tung. Similarity search on Bregman divergence: Towards non-metric indexing. *PVLDB*, 2(1):13–24, 2009.

[31] Tomás Skopal and Benjamin Bustos. On nonmetric similarity search problems in complex domains. *ACM Comput. Surv.*, 43(4):34:1–34:50, 2011.

[32] Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *ACM PODS*, pages 102–113, 2001.

[33] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.

[34] Charu C. Aggarwal, editor. *Managing and Mining Uncertain Data*. Springer, 2009.

[35] Anish Das Sarma, Omar Benjelloun, Alon Y. Halevy, and Jennifer Widom. Working models for uncertain data. In *ICDE*, page 7, 2006.

[36] Nilesh N. Dalvi and Dan Suciu. Management of probabilistic data: foundations and challenges. In *ACM PODS*, pages 1–12, 2007.

[37] Lei Chen and Xiang Lian. *Query Processing over Uncertain Databases*. Morgan & Claypool Publishers, 2012.

[38] Yiping Li, Jianwen Chen, and Ling Feng. Dealing with uncertainty: A survey of theories and practices. *IEEE TKDE*, 25(11):2463–2482, 2013.

[39] Jose Galindo. *Fuzzy Databases: Modeling, Design, and Implementation*. Idea Group Publishing, 2006.

[40] Suk Kyoon Lee. An extended relational database model for uncertain and imprecise information. In *VLDB*, pages 211–220, 1992.

[41] Amol Deshpande and Sunita Sarawagi. Probabilistic graphical models and their role in databases. In *VLDB*, pages 1435–1436, 2007.

[42] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[43] Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[44] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Evaluating probabilistic queries over imprecise data. In *ACM SIGMOD*, pages 551–562, 2003.

[45] Tingting Dong, Chuan Xiao, Xi Guo, and Yoshiharu Ishikawa. Processing probabilistic range queries over Gaussian-based uncertain data. In *SSTD*, pages 410–428, 2013.

[46] Serge Abiteboul, Paris C. Kanellakis, and Gösta Grahne. On the representation and querying of sets of possible worlds. *Theor. Comput. Sci.*, 78(1):158–187, 1991.

[47] Hans-Peter Kriegel, Peter Kunath, Martin Pfeifle, and Matthias Renz. Probabilistic similarity join on uncertain data. In *DASFAA*, pages 295–309, 2006.

[48] Christian Böhm, Frank Fiedler, Annahita Oswald, Claudia Plant, and Bianca Wackersreuther. Probabilistic skyline queries. In *CIKM*, pages 651–660, 2009.

[49] Goce Trajcevski, Roberto Tamassia, Hui Ding, Peter Scheuermann, and Isabel F. Cruz. Continuous probabilistic nearest-neighbor queries for uncertain trajectories. In *EDBT*, pages 874–885, 2009.

[50] Kai Zheng, Goce Trajcevski, Xiaofang Zhou, and Peter Scheuermann. Probabilistic range queries for uncertain trajectories on road networks. In *EDBT*, pages 283–294, 2011.

[51] Bruce S. E. Chung, Wang-Chien Lee, and Arbee L. P. Chen. Processing probabilistic spatio-temporal range queries over moving objects with uncertainty. In *EDBT*, pages 60–71, 2009.

[52] Yinuo Zhang, Anand V. Panangadan, and Viktor K. Prasanna. FP-CPNNQ: A filter-based protocol for continuous probabilistic nearest neighbor query. In *DASFAA*, pages 57–73, 2015.

[53] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffrey Scott Vitter. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*, pages 876–887, 2004.

[54] Christopher Re, Nilesh N. Dalvi, and Dan Suciu. Efficient top-k query evaluation on probabilistic data. In *ICDE*, pages 886–895, 2007.

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

[56] Avigdor Gal, Maria Vanina Martinez, Gerardo I. Simari, and V. S. Subrahmanian. Aggregate query answering under uncertain schema mappings. In *ICDE*, pages 940–951, 2009.

[57] Xiang Lian and Lei Chen. Efficient processing of probabilistic reverse nearest neighbor queries over uncertain data. *VLDB Journal*, 18(3):787–808, 2009.

[58] Mingwang Tang, Feifei Li, Jeff M. Phillips, and Jeffrey Jestes. Efficient threshold monitoring for distributed probabilistic data. In *ICDE*, pages 1120–1131, 2012.

[59] Bin Yang, Hua Lu, and Christian S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, pages 335–346, 2010.

[60] Aiden Zhijie Wang. Probabilistic range query over uncertain moving objects in constrained 2d space. *CoRR*, abs/1210.4663, 2012.

[61] Ming Hua and Jian Pei. Probabilistic path queries in road networks: traffic uncertainty aware path selection. In *EDBT*, pages 347–358, 2010.

[62] Xin Luna Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *VLDB*, pages 687–698, 2007.

[63] Thanh T. L. Tran, Liping Peng, Boduo Li, Yanlei Diao, and Anna Liu. PODS: a new model and processing algorithms for uncertain data streams. In *ACM SIGMOD*, pages 159–170, 2010.

[64] Andrew Nierman and H. V. Jagadish. Protdb: Probabilistic data in XML. In *VLDB*, pages 646–657, 2002.

[65] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Frequent subgraph pattern mining on uncertain graph data. In *CIKM*, pages 583–592, 2009.

[66] Amol Deshpande, Carlos Guestrin, Samuel Madden, Joseph M. Hellerstein, and Wei Hong. Model-based approximate querying in sensor networks. *VLDB Journal*, 14(4), 2005.

[67] Magnus Rattray, Xuejun Liu, Guido Sanguinetti, Marta Milo, and Neil D. Lawrence. Propagating uncertainty in microarray data analysis. *Briefings in Bioinformatics*, 7(1):37–47, 2006.

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

[69] Graham Cormode and Andrew McGregor. Approximation algorithms for clustering uncertain data. In *ACM PODS*, pages 191–200, 2008.

[70] Charu C. Aggarwal and Philip S. Yu. Outlier detection with uncertain data. In *SDM*, pages 483–493, 2008.

[71] Charu C. Aggarwal, Yan Li, Jianyong Wang, and Jing Wang. Frequent pattern mining with uncertain data. In *KDD*, pages 29–38, 2009.

[72] Simonas Šaltenis, Christian S. Jensen, Scott T. Leutenegger, and Mario A. Lopez. Indexing the positions of continuously moving objects. In *ACM SIGMOD*, pages 331–342, 2000.

[73] Kazuki Kodama, Tingting Dong, and Yoshiharu Ishikawa. An index structure for spatial range querying on Gaussian distributions. In *Proc. Fifth International Workshop on Management of Uncertain Data (MUD 2011)*, pages 1–7, 2011.

[74] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipies: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[75] Marios Hadjieleftheriou, Erik Hoel, and Vassilis J. Tsotras. Sail: A spatial index library for efficient application integration. *GeoInformatica*, 9(4):367–389, 2005.

[76] Reynold Cheng, Dmitri V. Kalashnikov, and Sunil Prabhakar. Querying imprecise data in moving object environments. *IEEE TKDE*, 16(9):1112–1127, 2004.

[77] Dieter Pfoser and Christian S. Jensen. Capturing the uncertainty of moving-object representations. In *Proc. 6th International Symposium on Advances in Spatial Databases (SSD'99)*, pages 111–132, 1999.

[78] Goce Trajcevski, Ouri Wolfson, Klaus Hinrichs, and Sam Chamberlain. Managing uncertainty in moving objects databases. *ACM TODS*, 29(3):463–507, 2004.

[79] Ouri Wolfson, A. Prasad Sistla, Sam Chamberlain, and Yelena Yesha. Updating and querying databases that track mobile units. *Distributed and Parallel Databases*, 7(3):257–287, 1999.

[80] Jinchuan Chen and Reynold Cheng. Efficient evaluation of imprecise location-dependent queries. In *ICDE*, pages 586–595, 2007.

[81] Kai Zheng, Xiaofang Zhou, Pui Cheong Fung, and Kexin Xie. Spatial query processing for fuzzy objects. *VLDB Journal*, 21(5):729–751, 2012.

[82] Xiang Lian and Lei Chen. A generic framework for handling uncertain data with local correlations. *PVLDB*, 4(1):12–21, 2010.

[83] Ying Zhang, Wenjie Zhang, Qianlu Lin, and Xuemin Lin. Effectively indexing the multi-dimensional uncertain objects for range searching. In *EDBT*, pages 504–515, 2012.

[84] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *ACM SIGMOD*, pages 322–331, 1990.

[85] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *ACM SIGMOD*, pages 47–57, 1984.

[86] Yannis Manolopoulos, Alexandros Nanopoulos, Apostolos N. Papadopoulos, and Yannis Theodoridis. *R-Trees: Theory and Applications*. Springer, 2005.

[87] Yufei Tao, Dimitris Papadias, and Jimeng Sun. The TPR*-tree: An optimized spatio-temporal access method for predictive queries. In *VLDB*, pages 790–801, 2003.

[88] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 2007.

[89] Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, 8(2):302–305, 2010.

[90] Pankaj K. Agarwal, Boris Aronov, Sariel Har-Peled, Jeff M. Phillips, Ke Yi, and Wuzhou Zhang. Nearest neighbor searching under uncertainty II. In *ACM PODS*, pages 115–126, 2013.

[91] George Beskales, Mohamed A. Soliman, and Ihab F. Ilyas. Efficient search for the top-k probable nearest neighbors in uncertain databases. *PVLDB*, 1(1):326–339, 2008.

[92] Sze Man Yuen, Yufei Tao, Xiaokui Xiao, Jian Pei, and Donghui Zhang. Superseding nearest neighbor search on uncertain spatial databases. *IEEE TKDE*, 22(7):1041–1055, 2010.

[93] Jeffrey Jestes, Graham Cormode, Feifei Li, and Ke Yi. Semantics of ranking queries for probabilistic data. *IEEE TKDE*, 23(12):1903–1917, 2011.

[94] Flip Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *ACM SIGMOD*, pages 201–212, 2000.

[95] Dimitris Papadias, Yufei Tao, Kyriakos Mouratidis, and Chun Kit Hui. Aggregate nearest neighbor queries in spatial databases. *ACM TODS*, 30(2):529–576, 2005.

[96] Yufei Tao, Dimitris Papadias, and Qiongmao Shen. Continuous nearest neighbor search. In *VLDB*, pages 287–298, 2002.

[97] Dirk Husmeier, Richard Dybowski, and Stephen Roberts. *Probabilistic Modelling in Bioinformatics and Medical Informatics*. Springer, 2005.

[98] Michael I. Mandel and Dan Ellis. Song-level features and support vector machines for music classification. In *Proc. 6th International Conference on Music Information Retrieval*, pages 594–599, 2005.

[99] Dominik Schnitzer, Arthur Flexer, and Gerhard Widmer. A filter-and-refine indexing method for fast similarity search in millions of music tracks. In *Proc. 10th International Conference on Music Information Retrieval*, pages 537–542, 2009.

[100] Minh N. Do and Martin Vetterli. Wavelet-based texture retrieval using generalized Gaussian density and Kullback-Leibler distance. *IEEE Trans. on Image Processing*, 11(2):146–158, 2002.

[101] Rajiv Mehrotra and James E. Gary. Feature-based retrieval of similar shapes. In *ICDE*, pages 108–115, 1993.

[102] Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *ACM SIGMOD*, pages 419–429, 1994.

[103] A.-V. I. Rosti. *Linear Gaussian models for speech recognition*. PhD thesis, Cambridge University, 2004.

[104] Chiyomi Miyajima, Yoshihiro Nishiwaki, Koji Ozawa, Toshihiro Wakita, Katsunobu Itou, Kazuya Takeda, and Fumitada Itakura. Driver modeling based on driving behavior and its evaluation in driver identification. *Proc. IEEE*, 95(2):427–437, 2007.

[105] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[106] Javier E. Contreras-Reyes and Reinaldo Boris Arellano-Valle. Kullback-Leibler divergence measure for multivariate skew-normal distributions. *Entropy*, 14(9):1606–1626, 2012.

[107] Paolo Ciaccia, Marco Patella, and Pavel Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *VLDB*, pages 426–435, 1997.

[108] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM TODS*, 30(1):41–82, 2005.

[109] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM TODS*, 30(2):364–397, 2005.

[110] Will D. Penny. KL-divergences of Normal, Gamma, Dirichlet and Wishart densities. Technical report, Wellcome Department of Cognitive Neurology, University College London, 2001.

[111] Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):11:1–11:58, 2008.

[112] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749, 2005.

[113] Mohammed J. Zaki and Wagner Meira, Jr., editors. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, 2014.

[114] I. N. Bronshtein and K. A. Semendyayev, editors. *A guide book to mathematics: fundamental formulas, tables, graphs, methods*. Springer, 1973.

List of Publications

Journal Papers

- Tingting Dong, Chuan Xiao, and Yoshiharu Ishikawa, “Probabilistic Range Querying over Gaussian Objects”, *The Institute of Electronics, Information and Communication Engineers (IEICE) Transactions on Information and Systems*, Vol. E97-D, No. 4, pp. 694–704, 2014.
- Tingting Dong, Yoshiharu Ishikawa, and Chuan Xiao, “Top-*k* Similarity Search over Gaussian Distributions Based on KL-Divergence”, *Journal of Information Processing (JIP)*, Vol. 24, No. 1, pp. 152–163, 2016.
- Tingting Dong, Yoshiharu Ishikawa, Chuan Xiao, and Jing Zhao, “*k*-Expected Nearest Neighbor Search over Gaussian Objects”, *Journal of Computers (JCP)*, Vol. 12, No. 2, pp. 105–115, 2017.

International Conference/Workshop Papers

- Kazuki Kodama, Tingting Dong, and Yoshiharu Ishikawa, “An Index Structure for Spatial Range Querying on Gaussian Distributions”, *Proceedings of the Fifth International Workshop on Management of Uncertain Data (MUD 2011), in conjunction with VLDB 2011*, pp. 1–7, Seattle, USA, August 2011.
- Masanori Mano, Xi Guo, Tingting Dong, and Yoshiharu Ishikawa, “Privacy Preservation for Location-Based Services Based on Attribute Visibility”, *Proceedings of*

the Second International Workshop on Information Management in Mobile Applications (IMMoA 2012), in conjunction with VLDB 2012, pp. 33–41, Istanbul, Turkey, August 2012.

- Tingting Dong, Chuan Xiao, Xi Guo, and Yoshiharu Ishikawa, "Processing Probabilistic Range Queries over Gaussian-based Uncertain Data", *Proceedings of the 13th International Symposium on Spatial and Temporal Databases (SSTD 2013)*, LNCS 8098, pp. 410–428, Munich, Germany, August 2013.

- Kento Sugiura, Arata Hayashi, Tingting Dong, and Yoshiharu Ishikawa, "Monitoring Query Processing in Mobile Robot Databases", *Proceedings of the Third International Workshop on Spatial Information Modeling, Management and Mining (SIM^3 2014), in conjunction with DASFAA 2014*, LNCS 8505, pp. 271–282, Bali, Indonesia, April 2014.

- Renhe Jiang, Jing Zhao, Tingting Dong, Yoshiharu Ishikawa, Chuan Xiao, and Yuya Sasaki, "A Density-based Approach for Mining Movement Patterns from Semantic Trajectories", *The IEEE Region 10 Conference (TENCON 2015)*, Macau, China, November 2015.

- Tingting Dong, Yoshiharu Ishikawa, Chuan Xiao, and Jing Zhao, "k-Expected Nearest Neighbor Search over Gaussian Objects", *The 4th International Conference on Network and Computing Technology (ICNCT 2015)*, Rome, Italy, December 2015.

国内学会発表（査読なし）

- 董 ていてい, 郭 茜, 石川 佳治, 肖 川, 「ガウス分布に対する確率的範囲問合せのための索引手法」, 第 4 回データ工学と情報マネジメントに関するフォーラム（DEIM 2012）, F11-4, 2012 年 3 月.
- 早矢仕 新, 董 ていてい, 加藤 翔, 石川 佳治, 「移動ロボットのための確率的空間問合せシステムの構築」, 情報処理学会第 74 回全国大会, 1T-8, 2012 年 3 月.
- 加藤 翔, 董 ていてい, 早矢仕 新, 石川 佳治, 「確率的な位置情報に基づくイベント問合せ」, 情報処理学会第 74 回全国大会, 4N-3, 2012 年 3 月.
- Tingting Dong, Xi Guo, Yoshiharu Ishikawa, and Chuan Xiao, “Indexing Gaussian Objects for Probabilistic Range Queries ” , *The 4th International Workshop with Mentors on Databases, Web and Information Management for Young Researchers (iDB 2012)*, August 2012.
- 董 テイテイ, 郭 茜, 肖 川, 石川 佳治, 「ガウス分布に対する確率的範囲問合せのための索引手法の評価」, 第 5 回データ工学と情報マネジメントに関するフォーラム（DEIM 2013）, A9-3, 2013 年 3 月. （学生プレゼンテーション賞受賞）
- 杉浦 健人, 早矢仕 新, 董 ていてい, 石川 佳治, 「移動ロボットデータベースにおけるモニタリング問合せ処理手法」, 情報処理学会第 75 回全国大会, 1P-1, 2013 年 3 月. （学生奨励賞受賞）
- 董 テイテイ, 肖 川, 石川 佳治, 「ガウス分布の類似問合せに関する考察」, 情報処理学会研究報告, Vol. 2013-DBS-157, No. 32 / Vol. 2013-IFAT-111, No. 32, 2013 年 7 月.
- 早矢仕 新, 杉浦 健人, 董 ていてい, 石川 佳治, 「曖昧な移動軌跡に対する範囲問合せ」, 第 12 回情報科学技術フォーラム（FIT 2013）, D-018, 2013 年 9 月.
- 早矢仕 新, 杉浦 健人, 董 ていてい, 石川 佳治, 「パーティクル表現を用いた曖昧位置情報に対する空間問合せ処理」, 第 6 回データ工学と情報マネジメントに関するフォーラム（DEIM 2014）, E4-6, 2014 年 3 月. （学生プレゼンテーション賞受賞）

- 趙 セイ, 董 テイテイ, 石川 佳治,「参加型センシングにおけるプライバシー保護手法」, 情報処理学会第 76 回全国大会, 5N-7, 2014 年 3 月.

- 董 テイテイ, 石川 佳治, 肖 川「KL 情報量に基づいたガウス分布の類似検索」, 第 7 回データ工学と情報マネジメントに関するフォーラム（DEIM 2015）, A6-3, 2015 年 3 月.

- 趙 菁, 姜 仁河, 董 テイテイ, 佐々木 勇和, 石川 佳治,「参加型センシングのためのタスク割当て手法」, 第 7 回データ工学と情報マネジメントに関するフォーラム（DEIM 2015）, C6-5, 2015 年 3 月.

- 姜 仁河, 趙 菁, 董 テイテイ, 佐々木 勇和, 石川 佳治,「密度に基づく意味的な軌跡パターンの発見」, 第 7 回データ工学と情報マネジメントに関するフォーラム（DEIM 2015）, E8-3, 2015 年 3 月.

- 趙 セイ, 石川 佳治, 肖 川, 董 テイテイ, 佐々木 勇和,「空間クラウドソーシングのための多様性を考慮したタスク割り当て手法」, 情報処理学会研究報告, 2015-DBS-161(8), 2015 年 8 月.

- 石川 佳治, 王 元元, 董 テイテイ, 杉浦 健人, 佐々木 勇和,「シミュレーションデータの分析管理のためのデータウェアハウスについて」, 第 14 回情報科学技術フォーラム（FIT 2015）, 3D-2, 2015 年 9 月.