k-Expected Nearest Neighbor Search over Gaussian Objects

Tingting Dong[1*], Yoshiharu Ishikawa[1], Chuan Xiao[2], Jing Zhao[1]

[1] Department of Systems and Social Informatics, Nagoya University, Nagoya, Aichi, Japan.
[2] Institute for Advanced Research, Nagoya University, Nagoya, Aichi, Japan.

* Corresponding author. Tel.: +81-052-789-3812; email: dongtt@db.ss.is.nagoya-u.ac.jp
Manuscript submitted September 8, 2015; accepted December 28, 2015.
doi: 10.17706/jcp.12.2.105-115

Abstract: *Probabilistic location information* has been attracting more and more attention due to the advances in computing devices and technologies, and has become an important research topic in recent years. In particular, *Gaussian distribution* is frequently used to represent probabilistic location information. On the other hand, 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 various areas. There have been considerable efforts made to extend nearest neighbor search over traditional location information to probabilistic location information. An example is the *expected distance*, which defines the distance over probabilistic location information. Following this trend, in this paper, we assume that the closeness between objects represented by Gaussian distributions are measured by their expected distance and consider the problem of *k-expected nearest neighbor search*. We analyze properties of expected distance on Gaussian distributions mathematically and derive its lower bound and upper bound. Based on our analysis, we propose three novel approaches to efficiently solve this problem. The efficiency of our approaches is demonstrated through extensive experiments.

Key words: Probabilistic location information, Gaussian distributions, nearest neighbor search, expected distance.

1. Introduction

In recent years, the advances in computing devices and technologies have been calling for researchers' attention 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 database area, there has been a large amount of work on representing uncertain location information by probabilistic models and proposing efficient solutions for query processing. A survey on this topic is given in [1]. 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 [2], [3].

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 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 [4], [5], which defines the distance over probabilistic location information. Following this trend, in this paper, we represent probabilistic location information by Gaussian distributions and assume that the closeness between Gaussian objects are measured by their expected *squared Euclidean distance* [5], 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 [6], [7], 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 top *k* objects having smallest distances with *q*. As shown in Fig. 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 [7]. The restaurant or shop may consider sending coupons or notifications to a number of nearest users for promotion.

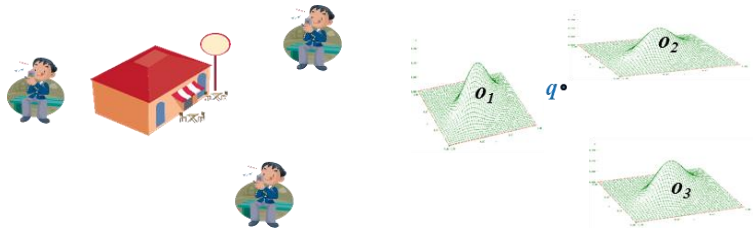


Fig. 1. An example.

The expected distance used in this paper 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 distributions mathematically and derive 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 distances of candidate objects without computing their actual distances. In refinement, we compute the actual distance for each candidate object 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 distributions 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 the paper is organized as follows. We formally define the problem in Section 2. Then in Section 3 we analyze expected distance on Gaussian distributions and derive its lower bound and upper bound. We propose three approaches in Section 4. Experimental results and analyses are presented in Section 5. We review related work in Section 6. Finally, Section 7 concludes the paper.

2. Problem Definition

In this paper, 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

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

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

$$ESQED(o,q) = \int \|x-q\|^2 \cdot p(x)dx. \quad (2)$$

We formally define the problem as follows.

Definition 1. *Given a database of Gaussian objects D = {o_1, …, o_n}, a query point q, a constant k, the problem of* ***k-Expected Nearest Neighbor Search over Gaussian objects*** *searches for k Gaussian objects that are nearest to q in D based on (2).*

The distance in (2) cannot be computed directly and is normally evaluated by numerical integration using the Monte Carlo methods. We employ the *importance sampling* [8] 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 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) and develop effective filtering algorithms (Section 4).

3. Lower Bound and Upper Bound

In [5], the following lemma is proved.

Lemma 1. *Given an object p with f(x) as its probability density function and $\bar{p}$ as its centroid, for any point q we have*

$$\int \|p-q\|^2 \cdot f(x)dx = \|\bar{p}-q\|^2 + \int \|x-\bar{p}\|^2 \cdot f(x)dx. \tag{3}$$

Based on this lemma, we can easily obtain

$$ESQED(o,q) = \|\mu-q\|^2 + \int \|x-\mu\|^2 \cdot p(x)dx. \tag{4}$$

In the subsequent discussion, we denote the second part as Δ2, i.e.,

$$\Delta^2 = \int \|x-\mu\|^2 \cdot p(x)dx, \tag{5}$$

and derive its lower bound and upper bound. According to [9], the lower bound and upper bound of Gaussian distribution is as follows:

$$\begin{aligned} p^{\perp}(x) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{\|x-\mu\|^2}{2\lambda^{\perp}}\right] \\ p^{\mathrm{T}}(x) &= \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{\|x-\mu\|^2}{2\lambda^{\mathrm{T}}}\right] \end{aligned} \tag{6}$$

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

$$\begin{aligned} \Delta^2 &\geq \int \|x-\mu\|^2 \cdot \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{\|x-\mu\|^2}{2\lambda^{\perp}}\right] dx \\ &= \frac{d \cdot (\lambda^{\perp})^{1+d/2}}{|\Sigma|^{1/2}} = \Delta^2_{\min} \end{aligned} \tag{7}$$

and its upper bound $\Delta^2_{\max}$ is

$$\begin{aligned} \Delta^2 &\leq \int \|x-\mu\|^2 \cdot \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{\|x-\mu\|^2}{2\lambda^{\mathrm{T}}}\right] dx \\ &= \frac{d \cdot (\lambda^{\mathrm{T}})^{1+d/2}}{|\Sigma|^{1/2}} = \Delta^2_{\max}. \end{aligned} \tag{8}$$

For the proof, please refer to [10].

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

$$LB(o,q) = \|\mu-q\|^2 + \Delta^2_{\min} \leq ESQED(o,q) \leq \|\mu-q\|^2 + \Delta^2_{\max} = UB(o,q). \tag{9}$$

4. Proposed Approaches

In this section, we propose three approaches for efficient query processing using the *filter-and-refine* paradigm described in Section 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 $\Delta^2_{\min}$ and $\Delta^2_{\max}$ involve costly matrix decomposition but they do not depend on queries. Based on this observation, we consider precomputing $\Delta^2_{\min}$ and $\Delta^2_{\max}$ offline to speed up online query processing.

In the first approach, we first precompute $\Delta^2_{\min}$ and $\Delta^2_{\max}$ 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, $(\mu, \Delta^2_{\min})$ is the lower left coordinate and $(\mu, \Delta^2_{\max})$ 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 (9) 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 $[(\mu_{\min}, \Delta^2_{\min}), (\mu_{\max}, \Delta^2_{\max})]$, its lower bound and upper bound distance can be computed similarly using (9):

$$LB(N,q) = \|\mu - q\|^2_{min} + \Delta^2_{min} \le ESQED(N,q) \le \|\mu - q\|^2_{max} + \Delta^2_{max} = UB(N,q). \quad (10)$$

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

Table 1. An Example Dataset

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

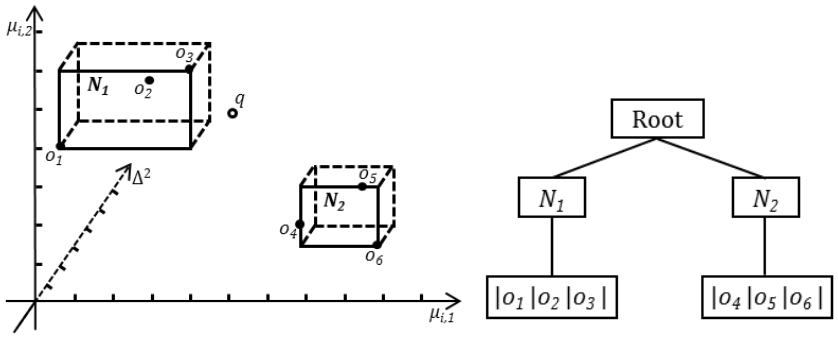


Fig. 2. R-tree image and structure of the example dataset.

In Fig. 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.

Fig. 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 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. Thus, we finish processing Q and continue to update the result set R using C.

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

Fig. 3. The **PLUB** algorithm.

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 $\Delta^2_{\min}$ for each Gaussian object, we insert each d-dimensional Gaussian object into an R-tree as a (d+1)-dimensional point (μ, $\Delta^2_{\min}$). We can easily compute the lower bound distance of each Gaussian object using (9) 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 (10).

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 7), 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 9, 17, and 20). In line 10, 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 22~26 and return C directly in Line 27.

Continuing with the example dataset in Table I 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. 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 third approach does not rely on precomputing and directly utilize the average of each Gaussian object, and is called **AVG**. Notice that by (9) we can obtain the least lower bound (**LLB**) of ESQED as follows:

$$LLB(o,q) = \|\mu - q\|^2 < \|\mu - q\|^2 + \Delta_{min}^2 \le ESQED(o,q). \tag{11}$$

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 μ. We compute the least lower bound distance of each Gaussian object using (11) 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.

Fig. 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. 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)}.

```
Input: R-tree, q, k
// Initialize the priority queue Q, the candidate set C
 1: Q ← Root of R-tree, C ← ∅
 2: while Q is not empty
 3:     N ←Q.top(), Q.pop()
 4:     if N is a leaf node
 5:         for each Gaussian object o_i in N
 6:             if C has less than k candidates
 7:                 insert o_i with ESQED(o_i, q) into C
 8:             else    // C has k or more than k candidates
 9:                 if LLB(o_i, q) is smaller than the k-th smallest ESQED in C
10:                     if ESQED(o_i, q) is smaller than the k-th smallest ESQED in C
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
14:             for each child entry N_i in N
15:                 insert N_i with LLB(N_i, q) into Q
16:         else    // C has k or more than k candidates
17:             for each child entry N_i in N
18:                 if LLB(N_i, q) is smaller than the k-th smallest ESQED in C
19:                     insert N_i with LLB(N_i, q) into Q
20:     if C has k candidates and
21:        LLB(Q.top(), q) is not smaller than the k-th smallest ESQED in C
22:         break
23: return C
```

Fig. 4. The **AVG** algorithm.

Table 2. Parameters for Testing

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

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.

5. Experiments

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 2.

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.

5.2. Effect of k

We show the average runtime and I/O access of the three approaches when varying k in Fig. 5(a) and Fig. 5(d). 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 that of **PLB**'s three-dimensional R-tree.

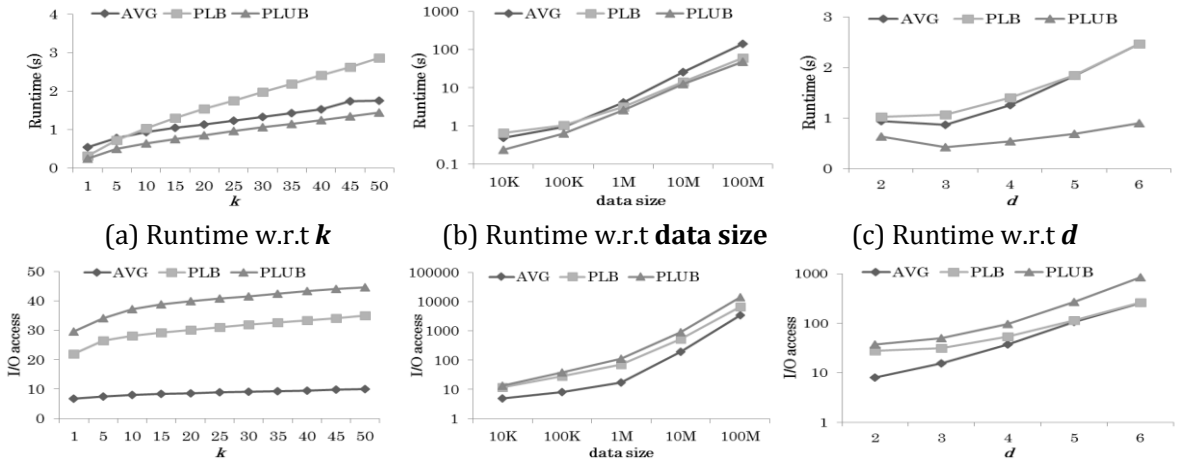
5.3. Effect of Data Size

As shown in Fig. 5(b) and Fig. 5(e), 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.

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. 5(c) and Fig. 5(f) 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 higher dimensions. This also indicates that **PLUB** is still best choice among the three approaches in terms of efficiency.



(a) Runtime w.r.t ***k*** (b) Runtime w.r.t **data size** (c) Runtime w.r.t ***d***

(d) I/O access w.r.t ***k*** (e) I/O access w.r.t **data size** (f) I/O access w.r.t ***d***

Fig. 5. Experimental results.

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**. 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**.

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 [11]. As uncertainty is inherent and immerging in many applications, there are a number of studies working on nearest neighbor searching under uncertainty [4], [5], [12], in which uncertainties are represented by probability distributions.

In [4], Ljosa *et al.* propose an approximate scheme to index *expected distance* for nearest neighbor search under the L_1 distance. Their result cannot be applied to our problem since we consider the exact case under the *squared Euclidean distance*. In [5], 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, [12] 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 [13], superseding nearest neighbor [14], and ranking queries [15].

However, none of the above work pays particular attention to Gaussian distribution, which is a popular probability distribution in many fields of applications. In [6], 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), [7] 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 paper.

Finally, it is worth mentioning that nearest neighbor search has numerous variations such as reverse NN search [16], aggregate NN search [17], continuous NN search [18], etc.

7. Conclusion

In this paper, 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 distributions and propose three novel approaches: **AVG**, **PLB**, and **PLUB**. We demonstrated the efficiency of our proposed approaches through extensive experiments and 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.

Acknowledgment

This research was partially supported by KAKENHI (25280039, 26540043).

References

[1] Wang, Y., Li, X., Li, X., & Wang, Y. (2013). A survey of queries over uncertain data. *Journal of Knowledge and Information Systems*, *37(3),* 485–530.

[2] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[3] Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern Classification* (2nd ed.). Wiley-Interscience.

[4] Ljosa, V., & Singh, A. K. (2007). APLA: Indexing arbitrary probability distributions. *Proceedings of the 23rd IEEE International Conference on Data Engineering* (pp. 946–955).

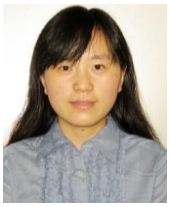
[5] Agarwal, P. K., Efrat, A., Sankararaman, S., & Zhang, W. (2012). Nearest-neighbor searching under uncertainty. *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 225–236).

[6] Böhm, C., Pryakhin, A., & Schubert, M. (2006). The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors. *Proceedings of the 22nd IEEE International Conference on Data Engineering* (p. 9).

[7] Dong, T., Xiao, C., & Ishikawa, Y. (2014). Probabilistic range querying over Gaussian objects. *The Institute of Electronics, Information and Communication Engineers Transactions, 4(97–D),* 694–704.

[8] Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.

[9] Ishikawa, Y., Iijima, Y., & Yu, J. X. (2009). Spatial range querying for Gaussian-based imprecise query objects. *Proceedings of the 25th IEEE International Conference on Data Engineering* (pp. 676–687).
[10] Nagoya. From http://www.db.ss.is.nagoya-u.ac.jp/~dongtt/references/JCP2015-Appendix.pdf
[11] Bhatia, N., & Vandana (2010). Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, *8(2),* 302–305.
[12] Agarwal, P. K., Boris, A., Har-Peled, S., Phillips, J. M., Yi, K., & Zhang, W. (2013). Nearest-neighbor searching under uncertainty II. *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 115–126).
[13] Beskales, G., Soliman, M. A., & Ilyas, I. F. (2008). Efficient search for the top-*k* probable nearest neighbors in uncertain databases. *Proceedings of the VLDB Endowment, 1(1),* 326–339.
[14] Yuen, S. M., Tao, Y., Xiao, X., Pei, J., & Zhang, D. (2010). Superseding nearest neighbor search on uncertain spatial databases. *IEEE Transactions on Knowledge and Data Engineering*, *22(7),* 1041–1055.
[15] Jestes, J., Cormode, G., Li, F., & Yi, K. (2011). Semantics of ranking queries for probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, *23(12),* 1903–1917.
[16] Korn, F., & Muthukrishnan, S. (2000). Influence sets based on reverse nearest neighbor queries. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (pp. 201–212).
[17] Papadias, D., Tao, Y., Mouratidis, K., & Hui, C. K. (2005). Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems, 30(2),* 529–576.
[18] Tao, Y., Papadias, D., & Shen, Q. (2002). Continuous nearest neighbor search. *Proceedings of the 28th International Conference on Very Large Data Bases* (pp. 287–298).



Tingting Dong is a PhD candidate in Graduate School of Information Science, Nagoya University, Japan. She received her MS degree in information science from Nagoya University, Japan in 2013, and both BS and BE degrees in mathematics and applied mathematics and software engineering from Dalian Jiaotong University, China in 2010. Her research interests include uncertain data management, spatio-temporal databases, and sensor databases.

Yoshiharu Ishikawa is a professor in Graduate School of Information Science, Nagoya University. His research interests include spatiotemporal databases, mobile databases, sensor databases, data mining, information retrieval, and Web information systems. He is a member of the Database Society of Japan, IPSJ, IEICE, JSAI, ACM, and IEEE.

Chuan Xiao is an assistant professor in Institute for Advanced Research, Nagoya University, Japan. He received bachelor's degree from Northeastern University, China in 2005, and PhD degree from The University of New South Wales, Australia in 2010. His research interests include similarity search, textual databases, and graph databases.

Jing Zhao is a PhD candidate in Graduate School of Information Science, Nagoya University, Japan. She received her MS degree in Information Science from Nagoya University, Japan in 2015, and BE degree in computer science & technology from Tianjin University of Science & Technology, China in 2012. Her research interests include spatial crowdsourcing, spatio-temporal databases, and data warehouse.