Finding Probabilistic Nearest Neighbors for Query Objects with Imprecise Locations

Yuichi Iijima
Graduate School of Information Science
Nagoya University
Nagoya, Japan
e-mail: iijima@db.itc.nagoya-u.ac.jp

Yoshiharu Ishikawa
Information Technology Center
Nagoya University
Nagoya, Japan
e-mail: ishikawa@itc.nagoya-u.ac.jp

***Abstract*—A nearest neighbor query is an important notion in spatial databases and moving object databases. In the emerging application fields of moving object technologies, such as mobile sensors and mobile robotics, the location of an object is often imprecise due to noise and estimation errors. We propose techniques for processing a nearest neighbor query when the location of the query object is specified by an imprecise Gaussian distribution. First, we consider two query processing strategies for pruning candidate objects, which can reduce the number of objects that require numerical integration for computing the qualification probabilities. In addition, we consider a hybrid approach that combines the two strategies. The performance of the proposed methods is evaluated using test data.**

I. Introduction

In recent years, the need for query processing technologies for objects which have *imprecise locations* is growing in the emerging application field of mobile databases. For example, in a mobile application, the use of a GPS system is a popular way to obtain the location of a moving object but, due to poor signal quality, an accurate location is not necessarily obtained [24]. In addition, frequent location measurements should be avoided when a GPS is powered by batteries. As a second example, consider an autonomous mobile robot which moves in the real-world. The robot continually estimates its own location based on sensor readings and movement histories using some statistical technique [29]. However, the estimated location is usually imprecise because of measurement noise. Third, the use of exact location information is prohibited in some types of mobile applications because of privacy issues. One solution is to add some noise to the exact location before providing it to the target application. In these examples, we can only obtain an imprecise location for a moving object. This means that traditional spatial query processing techniques should be enhanced by considering location impreciseness.

In this paper, we propose nearest neighbor query processing methods for the situation that a query object has an imprecise location and issues a nearest neighbor query. Specifically, we assume that the position of a query object is represented by a *Gaussian distribution*. The target of a query is a set of points which have static locations. We define a *probabilistic nearest neighbor query* by extending the traditional notion of a nearest neighbor query.

We introduce two query strategies for efficient query processing which are described in detail in Section IV. In addition, we propose a hybrid approach that combines the two strategies, and perform some experiments to compare the three strategies with different query settings. In [18], we proposed range query processing methods for a similar context. The proposed methods in this paper partly share the common ideas with this previous work, but the support for nearest neighbor queries requires development of new techniques. In particular, we use a Voronoi diagram for fast query processing.

The rest of this paper is organized as follows. In Section II, we describe related work. Section III introduces the idea of a probabilistic nearest neighbor query. Next, we describe the query processing techniques in Section IV. Section V shows the experimental results. Finally, conclusions are given in Section VI.

II. Related Work

Query processing techniques for imprecise information is a topic of some considerable interest in current database research [15], [23]. In mobile and spatio-temporal database research, imprecise location information appears due to variety of reasons such as noise in GPS signals or sensor readings [12] and location anonymity. Current proposals can be classified into the following types according to the situation: i) the locations of target objects are imprecise [9], [11], [21], [27], [28], ii), the location of a query object is imprecise [18], and iii) the locations of both the target objects and the query object are imprecise [5], [8], [19], [20]. This paper deals with the second case and assumes that the target objects are static points. Therefore, we can utilize conventional spatial index techniques [22], [26] such as R-trees for efficient query processing.

Another assumption is the *uncertainty model* used to represent the location impreciseness. The simplest approach is to assume that an object is located within a region (usually a rectangle is used) and its distribution obeys a uniform distribution [24]. On the other hand, some researchers allow the use of arbitrary probability distributions [5], [8], [13], [14], [21], [27], [28]. They usually assume that an *uncertainty region* is given for each object [8], [13], [14], [27], [28]. An uncertainty region represents the bounded area in which the object is located. The query processing approach often taken in this context is an extended version of the case for the uniform distribution.

In contrast, we consider location impreciseness based on Gaussian distributions. *Gaussian distributions* are a popular choice of probability density function (PDF) in statistics and pattern recognition [17]. In the context of moving object databases, Pfoser and Jensen described the idea of representing the location uncertainty of moving objects using Gaussian distributions [24]. Moreover, in robotics research, a Gaussian distribution is often used for *localization*, which is the process of estimating the location of a moving robot based on sensor readings and movement histories stored in the robot. *Kalman filters*, a popular localization technique, use the Gaussian distribution for modeling sensing noises, and the estimated location is represented by a Gaussian distribution. Because of its popularity, we focus on specialized techniques for Gaussian distributions in this paper. By effectively using the properties of Gaussian distributions, we can develop efficient query processing algorithms.

We can consider several types of queries based on imprecise locations such as range queries [11], [14], [18], [28], nearest neighbor queries [9], [11], [20], [21], join queries [19] and aggregation queries [10].

Cheng et al. [11] proposed a technique for nearest neighbor queries. The algorithm performs candidate pruning using the information of uncertainty regions. The technique is also applied to the context of moving object databases. [9] considered nearest neighbor queries for one-dimensional imprecise objects, and a type of query called *constrained nearest neighbor queries* is related to our work because it considers a given probability threshold. One feature of these is the presentation of an error bound for the result. Ljosa et al. *APLA* [21] present a method which constructs an approximated index for an arbitrary PDF. They proposed an approach for efficient k-nearest neighbor queries using APLA, but their definition of k-nearest neighbor queries is based on *expected distances*, which is different from our definition of nearest neighbors queries.

In all the approaches discussed so far, the target data objects are imprecise, but a given query is exact. There exist some proposals of nearest neighbor query processing methods for the case that both query and target objects are imprecise. Kriegel et al. proposed an approach that uses a sampling technique [20]. Beskalas et al. [5] recently proposed a processing method for k-nearest neighbor queries in which they consider uncertain regions like the other approaches, but also consider uncertainty of object existence. In addition, they try to strike a balance between the I/O process and the CPU process. However, none the methods discussed focus on Gaussian distributions.

In this paper, we extend our idea for probabilistic range queries [18] to probabilistic nearest neighbor queries. Although the current paper shares some common ideas with our previous work, the proposed methods consider the features of probabilistic nearest neighbor queries based on Gaussian distributions. For Gaussian distributions, Böhm et al. proposed an index structure called a *Gauss-tree* [6], [7]. It is used for indexing imprecise objects that obey Gaussian distributions. Since their subject is the impreciseness of target objects, this approach cannot be applied to our problem. In addition, their assumption is that the dimensions of the Gaussian distributions are independent, whereas our approach considers the correlation between dimensions.

III. Probabilistic Nearest Neighbor Queries

A *nearest neighbor query* aims to find the data object which is closest to the location of a given query object. If both data objects and the query object have exact locations, there is no problem; conventional query processing techniques can be applied. However, if either or both of them have imprecise locations, standard techniques cannot be applied. Since the result of a nearest neighbor query is determined by the positional relationship between the query object and target data objects, if the query object has an imprecise probabilistic location, the query result would be also probabilistic. In this paper, we deal with the situation that the imprecise location of the query object is specified by a Gaussian distribution.

For our purpose, it is necessary to expand the traditional nearest neighbor queries. We define a probabilistic nearest neighbor query formally as follows.

Definition 1: (**Probabilistic Nearest Neighbor Query**) *Assume that $\boldsymbol{x}$, the location of the query object q, is represented by a d-dimensional Gaussian distribution [17]*

$$p_q(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{q})^t\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{q})\right], \tag{1}$$

where $\boldsymbol{q}$ is the average of the distribution, $\boldsymbol{\Sigma}$ is a $d \times d$ covariance matrix, and $|\boldsymbol{\Sigma}|$ represents its determinant. Given the probability density function $p_q(\boldsymbol{x})$ and the probability threshold θ $(0 < \theta < 1)$, a probabilistic nearest neighbor query *$PNNQ(q,\theta)$ returns all the objects which satisfy the following condition: the probability that the object is the nearest neighbor of q (in terms of the Euclidean distance) is greater than or equal to θ. Let $\mathcal{O}$ be the set of data objects and let $\mathrm{Pr}_{NN}(o)$ be the probability that an object $o \in \mathcal{O}$ is the nearest neighbor of q:*

$$\mathrm{Pr}_{NN}(q,o) = \mathrm{Pr}(\forall o' \in \mathcal{O}, o' \neq o, \|\boldsymbol{x}-\boldsymbol{o}\|^2 \leq \|\boldsymbol{x}-\boldsymbol{o}'\|^2), \tag{2}$$

where $\boldsymbol{x}$ is the location of q and $\|\boldsymbol{x}-\boldsymbol{o}\|^2$ represents the squared Euclidean distance between $\boldsymbol{x}$, the location of the query object q, and $\boldsymbol{o}$, the location of the data object o. A probabilistic nearest neighbor query *is defined as follows:*

$$PNNQ(q,\theta) = \{n \mid n \in \mathcal{O}, \mathrm{Pr}_{NN}(q,n) \geq \theta\}. \tag{3}$$

Note that since $\boldsymbol{x}$, the location of q, is given probabilistically, an object can only be the nearest neighbor of q in a probabilistic sense. In contrast to the standard nearest neighbor query, a probabilistic nearest neighbor query can return zero or multiple objects. In the following section, we propose the query processing strategies for probabilistic nearest neighbor queries. Although our main target is query processing in a two-dimensional space, our algorithms are generic and can be applied to higher dimensions.

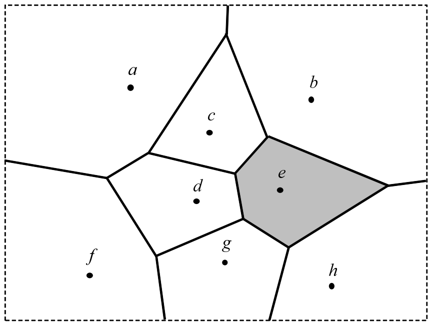

Fig. 1. Voronoi diagram

IV. Query Processing Methods

A. Basic Ideas

In the conventional non-imprecise setting, a nearest neighbor query can be answered efficiently using a *Voronoi diagram* [4], [16]. In this paper, we also utilize a Voronoi diagram for processing imprecise nearest neighbor queries. Given a set of points over a space, a Voronoi diagram is constructed by dividing the space according to which of the given points is the nearest. A Voronoi diagram can be constructed with $O(n \log n)$ time for the two-dimensional case and with $O(n^{\lfloor (d+1)/2 \rfloor})$ time for the d-dimensional case ($d \geq 3$). Figure 1 shows an example of a Voronoi diagram for eight point objects (a to h) over a two-dimensional plane. The influence area of each object is called a *Voronoi region*, and we denote the Voronoi region of object o by V_o. For example, the shaded area shown in Fig. 1 is Voronoi region, V_e, for object e.

Using a Voronoi diagram, we can reformulate the notion of a probabilistic nearest neighbor given in Eq. (2). For an object $o \in \mathcal{O}$, if the query object q is located in V_o, o becomes the nearest neighbor of q. Therefore, $\mathrm{Pr}_{NN}(q, o)$ can be computed by calculating the probability that the query object q is inside V_o. This means that we should integrate the probability density function $p_q(\boldsymbol{x})$ for the region V_o:

$$\mathrm{Pr}_{NN}(q, o) = \int_{\boldsymbol{x} \in V_o} p_q(\boldsymbol{x}) d\boldsymbol{x}. \tag{4}$$

Since we assume that the location of the query object is represented by a Gaussian distribution, computation of $\mathrm{Pr}_{NN}(\cdot)$ needs the integration of the Gaussian distribution given in Eq. (1) over V_o, the Voronoi region of the data object o. However, this requires costly numerical integration (e.g., using the Monte Carlo method) because it is impossible to compute the integral of the Gaussian PDF analytically. Moreover, the polyhedral shape of each Voronoi region increases the numerical integration cost. Therefore, if we compute $\mathrm{Pr}_{NN}(\cdot)$ for all data objects, an extremely high cost is incurred. To solve this problem, our approach prunes the non-candidate objects whose $\mathrm{Pr}_{NN}(\cdot)$ can be seen to be obviously smaller than the threshold θ without numerical integration. We propose two query processing strategies, which are described in Subsections IV-B and IV-C.

B. Strategy 1

In Strategy 1, we use the notion of an *uncertainty region* [27], [28]. This is a region in which the query object is located with a specified probability. In particular, we use the term *θ-region* [18] to denote the region for which the probability threshold is $1 - 2\theta$.

Definition 2: **($\boldsymbol{\theta}$-Region)**
Consider the integration of the probability density function $p_q(\boldsymbol{x})$ over an ellipsoidal region $(\boldsymbol{x} - \boldsymbol{q})^t \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{q}) \leq r^2$. Given θ $(0 < \theta < 1/2)$, let the value of r for which the result of the integration becomes $1 - 2\theta$ be r_θ:

$$\int_{(\boldsymbol{x}-\boldsymbol{q})^t \boldsymbol{\Sigma}^{-1} (\boldsymbol{x}-\boldsymbol{q}) \leq r_\theta^2} p_q(\boldsymbol{x}) d\boldsymbol{x} = 1 - 2\theta. \tag{5}$$

We call the ellipsoidal region

$$(\boldsymbol{x} - \boldsymbol{q})^t \boldsymbol{\Sigma}^{-1} (\boldsymbol{x} - \boldsymbol{q}) \leq r_\theta^2 \tag{6}$$

defined by r_θ the θ-region.

Since a θ-region is dependent on given parameters of the query, we need to derive it dynamically when a query is made. The simplest way for computing the θ-region for a given query is to perform a binary search to find an appropriate θ-value that satisfies Eq. (5), but this is not realistic. To cope with this problem, we transform the integration over an ellipsoidal region to an integration over a d-dimensional sphere. To begin with, let us introduce the *normalized Gaussian distribution* defined by assigning values $\boldsymbol{q} = \boldsymbol{0}$ and $\boldsymbol{\Sigma} = \mathbf{I}$ in Eq. (1).

$$p_{\mathrm{norm}}(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{0}, \mathbf{I}) = \frac{1}{(2\pi)^{d/2}} \exp\left[-\frac{1}{2}\|\boldsymbol{x}\|^2\right] \tag{7}$$

Based on this probability density function, we can derive the following property.

Property 1: Consider the integration of $p_{\mathrm{norm}}(\boldsymbol{x})$ over the region $\|\boldsymbol{x}\|^2 \leq r^2$, which is a sphere with the origin as its center and radius r. For a given θ $(0 < \theta < 1/2)$, let $\tilde{r}_\theta$ be the radius with which the integral becomes $1 - 2\theta$. Then

$$\int_{\|\boldsymbol{x}\|^2 \leq \tilde{r}_\theta^2} p_{\mathrm{norm}}(\boldsymbol{x}) d\boldsymbol{x} = 1 - 2\theta \tag{8}$$

and

$$r_\theta = \tilde{r}_\theta \ . \tag{9}$$

The proof is given in [18]. Thus, if $\tilde{r}_\theta$ is calculated for a given θ using Eq. (8), the equality in Eq. (9) means we can use it for our purpose.

Since the shape of the θ-region is ellipsoidal, it is not easy to directly apply it to pruning. Therefore, we derive the bounding box for the given θ-region as shown in Fig. 2. Let the width of the box from the average of the distribution $\boldsymbol{q}$ along the i-th dimension axis be w_i. Then the following property holds [18].

Property 2: The value of w_i $(i = 1, 2, \ldots, d)$ is given by

$$w_i = \sigma_i r_\theta, \tag{10}$$

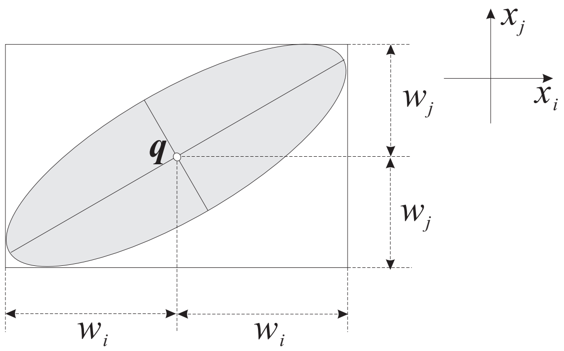

Fig. 2. Using bounding box

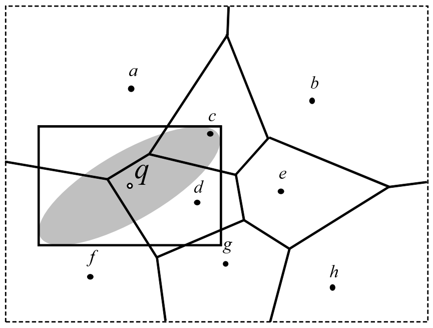

Fig. 3. Bounding box and Voronoi regions

where σ_i corresponds to the standard deviation for the i-th dimension. That is,

$$\sigma_i = \sqrt{(\mathbf{\Sigma})_{ii}}, \tag{11}$$

where $(\mathbf{\Sigma})_{ii}$ represents the (i,i) entry of $\mathbf{\Sigma}$.

Figure 3 illustrates the idea for query processing. The shaded area is the θ-region which is bounded by a box. The key point is that we should only consider the points whose Voronoi regions overlap with the bounding box. In this case, the points a, c, d, f, and g become the candidates. We can exclude the non-overlapped Voronoi regions from further consideration.

The reason is as follows. First, the probability that the query object is located outside of the bounding box is smaller than $1-(1-2\theta) = 2\theta$. Second, since a Gaussian distribution is point symmetric, if we draw the symmetry region V_o' for a Voronoi region V_o in terms of $\boldsymbol{q}$, the integral over V_o' gives the same probability as that over V_o. This means that the integral over V_o is less than θ so that $\Pr_{NN}(o)$ is never larger than or equal to θ. On the other hand, the remaining objects may satisfy the qualification condition.

The query processing strategy can be summarized as follows. First, we retrieve as the candidate objects those objects whose Voronoi region overlaps with the bounding box of the θ-region. Then, we compute $\Pr_{NN}(\cdot)$ of all candidate objects by numerical integration. Thus we do not need to compute $\Pr_{NN}(\cdot)$ for the objects whose integrals are obviously smaller than θ.

Algorithm 1 PNNQ Based on Strategy 1

```
1: procedure PNNQ-1($\boldsymbol{q}$, $\mathbf{\Sigma}$, $\theta$)
2:     $\mathcal{C} \leftarrow \emptyset$, $sum \leftarrow 0$
3:     Calculate $\sigma_i$ ($i = 1, \ldots, d$) from $\mathbf{\Sigma}$
4:     $r_\theta \leftarrow$ catalog_lookup($\theta$)
5:               ▷ may return the closest approximation value
6:     Using $\{\sigma_i\}_{i=1}^{d}, r_\theta$, derive the bounding box shown in Fig. 2
7:     Retrieve objects whose Voronoi region overlap with the bounding box into $\mathcal{C}$
8:     foreach $o \in \mathcal{C}$ do
9:         Compute $\Pr_{NN}(q, o) = \int_{\boldsymbol{x} \in V_o} p_q(\boldsymbol{x}) d\boldsymbol{x}$
10:        $sum \leftarrow sum + \Pr_{NN}(q, o)$
11:        if $\Pr_{NN}(q, o) \geq \theta$ then
12:            output $o$
13:        end if
14:        if $sum > 1 - \theta$ then
15:            return
16:        end if
17:    end for
18: end procedure
```

There exists one remaining issue for query processing: how to obtain r_θ from the given θ. Referring back to Property 1, we can obtain the θ-region (Eq. (6)) for the given θ by integrating the normalized formula Eq. (7) over a sphere. However, since the function $p_{\rm norm}(\boldsymbol{x})$ of Eq. (7) cannot be integrated analytically, it is not possible to compute r_θ directly from θ. To solve this problem, we use a table (constructed in advance by numerical integration) which contains θ and its corresponding r_θ for each representative value of r. Using the table, we can quickly find the θ-region. Such an idea is also used in [27], [28], and the table is called a *U-catalog*. We should note that the corresponding entry for the given θ-value may not exist in the table. For this case, we find the entry r_θ^* which is the maximal within the entries that satisfy $\theta^* < \theta$. Although the number of retrieved objects may increase, the correctness of the result is retained.

Based on the above considerations, the query processing algorithm is the procedure shown as Algorithm 1. Note that the following preparation is required in advance.

1) Create a U-catalog.
2) Compute a Voronoi diagram for the target data objects.
3) For each data object, store its coordinates and the information about the corresponding Voronoi region in a file.

The function catalog_lookup($\cdot$) at line 3 obtains the appropriate r_θ using the U-catalog. If the given θ does not match the entries, it returns the conservative approximation value r_θ^*. The algorithm is simple, but we have an additional improvement. If the condition at line 14 is satisfied, the remaining candidates cannot satisfy the threshold θ so that we can terminate the process here.

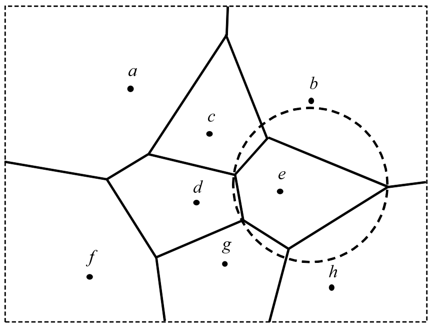

Fig. 4. Smallest enclosing circle of e's Voronoi region

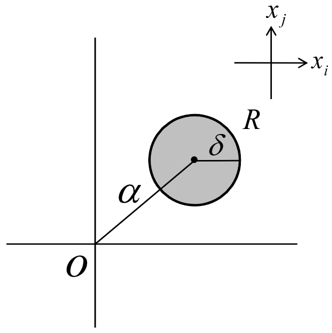

Fig. 5. Sphere R

α	δ	$\pi(\alpha, \delta)$
0.0	0.1	$\cdots$
0.0	0.2	$\cdots$
$\vdots$	$\vdots$	$\vdots$
1.0	0.1	$\cdots$
$\vdots$	$\vdots$	$\vdots$

Fig. 6. U-catalog for Strategy 2

C. Strategy 2

In Strategy 2, we perform pruning using the upper bound value of $\mathrm{P}_{NN}(\cdot)$ for each data object. First, we obtain the *smallest enclosing sphere* (*SES*) for each Voronoi region. For the two-dimensional case, the corresponding *smallest enclosing circle* [1] can be computed $O(m)$, where m is the number of vertices of the Voronoi region. For instance, Fig. 4 shows the SES for the Voronoi region V_e. If integrate the given probability density function $p_q(\boldsymbol{x})$ over the SES, the result can be used as an upper-bound for the target probability which we seek. The integral over a sphere can be estimated easily by creating a table in advance, as shown below. This technique is effective for the fast pruning of candidate objects. The approach is described in detail below. First, we consider a simple case in which the covariance matrix $\boldsymbol{\Sigma}$ in Eq. (1) is a unit matrix, and then we extend the idea to the general case.

1) Case of $\boldsymbol{\Sigma} = \mathbf{I}$*:* In this case, $p_q(\boldsymbol{x})$ is reduced to $p_{\mathrm{norm}}(\boldsymbol{x})$ in Eq. (7). The equi-probable surface of this PDF is spherical.

Now we consider how to use the information of SESs for efficient query processing. The sizes and the locations of the Voronoi regions for target objects are different from each other. Therefore, the radii and the coordinates of the respective centers of their SESs are also different. To obtain the integrals quickly for different SESs, we create a table in advance. Consider the normalized Gaussian distribution $p_{\mathrm{norm}}(\boldsymbol{x})$ given in Eq. (7). Suppose that there is a d-dimensional sphere R with radius δ and let the distance between its center and the origin be α as shown in Fig. 5. Now we consider the probability

$$\pi(\alpha, \delta) = \int_{\boldsymbol{x} \in R} p_{\mathrm{norm}}(\boldsymbol{x}) d\boldsymbol{x}, \qquad (12)$$

which is the result of the integration of $p_{\mathrm{norm}}(\boldsymbol{x})$ over the sphere R, represented by the two parameters α and δ. For different combinations of α and δ, we compute the probabilities using numerical integration and store the results in a table as shown in Fig. 6. We also call this table a *U-catalog*. Given a pair (α, δ), a U-catalog returns the corresponding probability.

Next, we describe the usage of a U-catalog for query processing. Suppose that a probabilistic nearest neighbor query $PNNQ(q, \theta)$ is issued. To evaluate whether each data object o may satisfy the query, we need compute the integral $\int_{\boldsymbol{x} \in SES_o} p_{\mathrm{norm}}(\boldsymbol{x}) d\boldsymbol{x}$, where SES_o is the SES for V_o, the Voronoi region of o. Since α_o, the distance between the center of SES_o and the origin, and δ_o, the radius of SES_o, can be easily obtained, we can search the U-catalog shown in Fig. 6 to find the entry (α_o, δ_o). If the corresponding probability is less than θ, we can safely prune the object o. On the other hand, even if the value is larger than or equal to θ, it is not assured that o satisfies the query because the bounding sphere SES_o has a large volume compared to V_o. Therefore, in this case, it is necessary to compute the integration over the Voronoi region numerically.

As in the case of Strategy 1, since the number of entries in a U-table is limited, it may not contain the entry for the given (α_o, δ_o). In that case, we find the entry (α_o^*, δ_o^*) whose probability is the smallest within the entries that satisfy $\alpha_o^* \leq \alpha_o$ and $\delta_o^* \geq \delta_o$. That is to say, we find the closest entry under the condition that the probability is larger than the actual value. Although we may need to process additional candidate objects, the approach assures the correctness of the process. Next, we generalize this idea and then present a query processing algorithm.

2) General Case: Now we consider the case that $\boldsymbol{\Sigma}$ in Eq. (1) is an arbitrary covariance matrix. Since the equi-probable surface of $p_q(\boldsymbol{x})$ has an ellipsoidal shape, it is not possible to obtain the integral of $p_q(\boldsymbol{x})$ over the SES of each Voronoi region by looking up a U-catalog. Therefore, we introduce $p_q^\top(\boldsymbol{x})$, the upper-bounding function of $p_q(\boldsymbol{x})$.

Definition 3: (**Upper-bounding Function**)
Let the spectral decomposition of the inverse of the covariance matrix $\boldsymbol{\Sigma}^{-1}$ *be*

$$\boldsymbol{\Sigma}^{-1} = \sum_{i=1}^{d} \lambda_i \boldsymbol{v}_i \boldsymbol{v}_i^t, \qquad (13)$$

where λ_i *and* $\boldsymbol{v}_i$ *are the i-th eigenvalue and its corresponding eigenvector. Let us define*

$$\lambda^\top = \min\{\lambda_i\}. \qquad (14)$$

Note that $\lambda^\top > 0$ *holds because all the eigenvalues of a covariance matrix are greater than zero. We define the matrix*

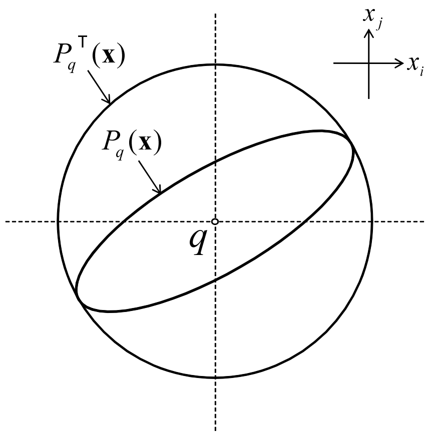

Fig. 7. $p_q(\boldsymbol{x})$ and $p_q^\top(\boldsymbol{x})$

$\mathbf{M}^\top$ *as follows:*

$$\mathbf{M}^\top = \lambda^\top \sum_{i=1}^{d} \boldsymbol{v}_i \boldsymbol{v}_i^t = \lambda^\top \mathbf{I}. \qquad (15)$$

Then, we define the following function obtained by substituting $\boldsymbol{\Sigma}^{-1}$ *in Eq. (1) with* $\mathbf{M}^\top$*:*

$$p_q^\top(\boldsymbol{x}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{\lambda^\top}{2}\|\boldsymbol{x}-\boldsymbol{q}\|^2\right]. \qquad (16)$$

The equi-probable surface of $p_q^\top(\boldsymbol{x})$ is spherical. Note that $p_q^\top(\boldsymbol{x})$ is *not* a probability density function since its integral over the whole space is not equal to one. The function $p_q^\top(\boldsymbol{x})$ has the following property:

Property 3: For any point $\boldsymbol{x}$*,*

$$p_q(\boldsymbol{x}) \le p_q^\top(\boldsymbol{x}). \qquad (17)$$

$p_q^\top(\boldsymbol{x})$ is the function with the smallest spherical equi-probable surface that satisfies this property. This means that $p_q^\top(\boldsymbol{x})$ is the least upper-bounding function of $p_q(\boldsymbol{x})$. Figure 7 illustrates $p_q^\top(\boldsymbol{x})$. This figure shows the isosurface of equal probability for each function.

It is easy to obtain the integral $p_q^\top(\boldsymbol{x})$ by looking in the U-catalog constructed for the case of $\boldsymbol{\Sigma} = \mathbf{I}$. Since $p_q^\top(\boldsymbol{x})$ satisfies Property 3, the integral of $p_q^\top(\boldsymbol{x})$ is never smaller than that of $p_q(\boldsymbol{x})$ for the same integration region. Therefore, the obtained value can be used for pruning: if the estimated integral of $p_q^\top(\boldsymbol{x})$ is smaller than θ, the object does not have any chance of satisfying the query condition. Note that if there is no entry that exactly matches the given parameters (α, δ), we should find the closest conservative setting as described for the case of $\boldsymbol{\Sigma} = \mathbf{I}$.

The query processing algorithm based on Strategy 2 is given as Algorithm 2. Note that the following preprocessing is required in advance.

1) Construct a U-catalog.
2) Compute Voronoi regions and their SESs for all data objects.
3) For each data object, store its coordinates, the center, and the radius of its SES, and information about its Voronoi region in a file.

Algorithm 2 PNNQ Based on Strategy 2

```
1: procedure PNNQ-2($\boldsymbol{q}$, $\boldsymbol{\Sigma}$, $\theta$)
2:     $\mathcal{C} \leftarrow \emptyset$, $sum \leftarrow 0$
3:     Calculate $\lambda^\top$ and $|\boldsymbol{\Sigma}|$ from $\boldsymbol{\Sigma}$
4:     for each data object $o \in \mathcal{O}$ do
5:         Calculate the distance $\alpha_o$
6:         $\pi(\alpha_o, \delta_o) \leftarrow$ catalog_lookup$(\alpha_o, \delta_o)$
7:             ▷ may return the closest approximation value
8:         if $\pi(\alpha_o, \delta_o) \ge \theta$ then
9:             $\mathcal{C} \leftarrow \mathcal{C} \cup \{o\}$
10:        end if
11:    end for
12:    Sort objects in $\mathcal{C}$ in descending order of $\pi(\alpha, \delta)$
13:    for each $o \in \mathcal{C}$ do            ▷ with descending order
14:        Compute $\Pr_{NN}(o) \leftarrow \int_{\boldsymbol{x} \in V_o} p_q(\boldsymbol{x}) d\boldsymbol{x}$
15:        $sum \leftarrow sum + \Pr_{NN}(o)$
16:        if $\Pr_{NN}(o) \ge \theta$ then
17:            output $o$
18:        end if
19:        if $sum > 1 - \theta$ then
20:            return
21:        end if
22:    end for
23: end procedure
```

The function catalog_lookup at line 6 finds the corresponding entry for the parameters (α_o, δ_o), where α_o is the distance between the center of SES_o and the query center $\boldsymbol{q}$ and δ_o is the radius of SES_o. The function returns the probability for SES_o. If it cannot find an exact match, it returns the closest value as described above. The reason why we sort objects in descending order of the probability (line 12) is for efficiency. If this optimization is applied, the query processing terminates more quickly on average than is the case for a random order, because an object which has a large probability for its SES usually also has a large probability over its Voronoi region.

V. Experiments

A. Experimental Setup

For the experiments, we used road line segment data of Long Beach from the TIGER database [3]. We extracted the midpoint for each line segment and made a point set. This set consisted of 50,747 points and was normalized in $[0, 1000] \times [0, 1000]$ space. We evaluated the two query strategies proposed in Section IV for $PNNQ(q, \theta)$ using this dataset. We also evaluated their combination. This hybrid strategy performs filtering using Strategy 1, then does additional filtering using Strategy 2 for the remaining objects.

The covariance matrix $\boldsymbol{\Sigma}$ in Eq. (1) was defined to be

$$\boldsymbol{\Sigma} = \gamma \begin{bmatrix} 7 & 2\sqrt{3} \\ 2\sqrt{3} & 3 \end{bmatrix}$$

For this matrix the shape of the isosurface of $p_q(\boldsymbol{x})$ is an ellipse titled at 30° and the major-to-minor axis ratio is 3:1. The coefficient γ specifies the overall uncertainty of the distribution. We used $\gamma = 10$ and $\theta = 0.01$ as the default but we performed evaluations with different γ and θ values. Moreover, we also evaluated strategies for different isosurface shapes by changing elements of $\boldsymbol{\Sigma}$. We used the averaged response time of ten query trials as the evaluation metric.

We used LEDA 6.1 for computing Voronoi regions and smallest enclosing circles. LEDA [1] is a C++ class library with geometric data types and algorithms, and is based on deep knowledge of graph and network problems and computational geometry. In addition, we used RANDLIB [2] to compute probabilities by numerical integration. RANDLIB is a C library which can be used for generating random numbers that obey a specified two-dimensional Gaussian distribution. The proportion of such random numbers that fall in the specified region corresponds to the probability to be estimated. This method is called the *importance sampling* technique [25]. It is a kind of Monte Carlo method, but is more efficient than the standard Monte Carlo approach. In these experiments, 1,000,000 random numbers were generated for each numerical integration procedure. It took about 0.8 seconds for the numerical integration for one object.

The programs for the experiments were implemented using C++ and were run on a PC with an Intel Pentium CPU (2.0 GHz), 1GB of memory, a 143GB hard disk, and Fedora Core 5 OS.

B. Experimental Results

1) Experiments Using Default Parameters: We show the query processing time for each strategy with default parameter settings ($\gamma = 10, \theta = 0.01$) in Fig. 8. We also show the number of candidate objects that required numerical integration in Table I. The number of resulting answer objects is 26.

Figure 8 shows that most of the processing time is spent in numerical integration for deriving $\Pr_{NN}(\cdot)$. Since our method reduces the computational cost by pruning objects, a strategy that can prune more objects will have better performance. Table I shows that the number of objects that require numerical integration is 177 in Strategy 1 and 157 in Strategy 2 and the hybrid approach reduces the number to 132. The reason is that the candidate sets obtained by the two strategies share common objects. The hybrid strategy only needs to perform numerical integration for the common objects.

The reader should note that the response time is quite large even for the hybrid strategy; it takes nearly 120 seconds for obtaining only 26 answer objects. To reduce the processing time further, the most effective way is to reduce the time for numerical integration. One obvious approach is to use a smaller number of random numbers for the numerical integration procedure. For example, if we use 100,000 random numbers instead of 1,000,000 in this experiment, the integration time is approximately reduced to $1/10$ of what it was and we can achieve efficient query processing. However, this approach has a negative effect on accuracy. In this experiment, we use the

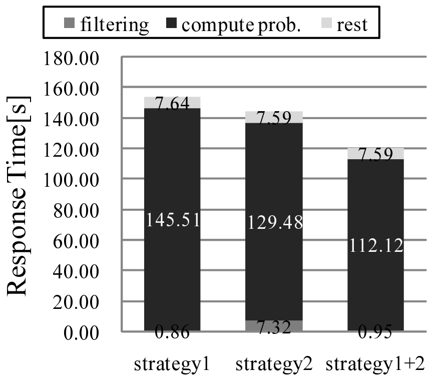

Fig. 8. Response time ($\gamma = 10, \theta = 0.01$)

TABLE I
NUMBER OF CANDIDATES ($\gamma = 10, \theta = 0.01$)

Strategy 1	Strategy 2	Strategy 1+2
177	157	132

probability threshold value $\theta = 0.01$. This smaller parameter setting also requires accuracy for numerical integration so we chose to use 1,000,000 samples based on several trials. If the probability threshold value is larger, for example $\theta = 0.1$, we may be able to reduce the number of samples without loss of accuracy.

Figure 8 shows that the processing time for filtering in Strategy 2 is almost ten times larger than that in Strategy 1. The reason is that in lines 4 to 11 of Algorithm 2, we scan all the table entries. The processing time is improved in the hybrid strategy since it uses Strategy 1 first and then Strategy 2 only checks the selected candidate objects. Although we use a naive strategy for the lookup phase in Algorithm 2, we would be able to improve the performance of the algorithm using an additional index structure. For example, we could construct an in-memory spatial index for the entries of the U-catalog using the center of each SES as an index key. If we perform a nearest neighbor query from $\boldsymbol{q}$ using the index, we can arrange the entries in ascending order of the α values. Then we can check the promising entries first. We will evaluate this optimization method in future work.

Figures 9, 10, and 11 show the candidate objects in each strategy for a query with default parameters. The small white circle located in the center in each figure denotes the average point of the distribution. The polygons with thick lines and the black polygons correspond to the candidate objects and the result objects, respectively. The rectangles in Figs. 9 and 10 are the bounding boxes of the θ-regions used for filtering in Strategy 1.

2) Using Different γ Values ($\gamma = 1$ and $\gamma = 50$): To examine the impact of the degree of uncertainty on the performance of the strategies, we changed γ, which determines the uncertainty of the covariance matrix $\boldsymbol{\Sigma}$ in Eq. (1) to $\gamma = 1$

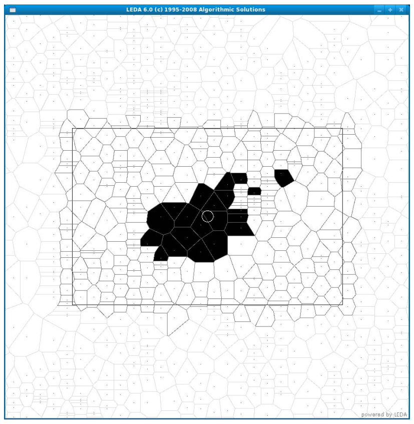

Fig. 9. Candidate objects in Strategy 1

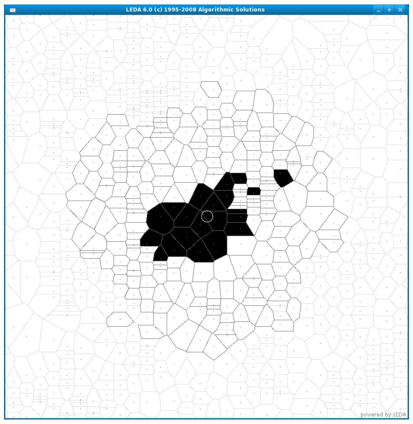

Fig. 10. Candidate objects in Strategy 2

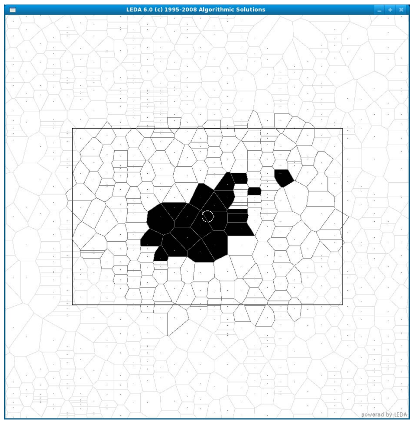

Fig. 11. Candidate objects in Strategy 1+2

and $\gamma = 50$. If we change γ, the shape of the isosurface of $p_q(\boldsymbol{x})$ does not change but its size does. Specifically, if γ becomes larger, the size of the isosurface also becomes larger. The size of the isosurface represents the degree of uncertainty of location of the query object. Therefore, compared to the default value $\gamma = 10$, $\gamma = 1$ means that the location of the query object is more accurately known while $\gamma = 50$ represents a more vague location.

Figures 12 and 13 show the query processing time for each of the strategies with $\gamma = 1$ and $\gamma = 50$, respectively. Tables II and III show the number of candidate objects that require numerical integration. The number of resulting objects is 8 in the case of $\gamma = 1$, and is 15 in the case of $\gamma = 50$.

Figure 12 shows that the proportion of the processing time spent on probability computation in the case of $\gamma = 1$ is lower than that of $\gamma = 10$. The reason is that the number of candidate objects is small because the location of the query object is not so uncertain. On the other hand, the processing time for filtering is unchanged by the degree of uncertainty of the query object. Therefore, in the case of $\gamma = 1$, since the ratio of the processing time for filtering is larger than that of $\gamma = 10$, Strategy 2 (which has a high filtering cost) has a longer processing time than Strategy 1.

We can see an interesting result in Fig. 12. In the case of $\gamma = 1$, since the number of the candidate objects in Strategy 2 is larger than that of Strategy 1, it would seem that the processing time for probability computation in Strategy 2 should take more time than Strategy 1. However, the result is the opposite. The reason is that Algorithm 2 performs an ordering of candidates at line 12. Since the promising objects are processed earlier, the query process can terminate more quickly.

In the case of $\gamma = 50$, when the location of the query object is more imprecise, the processing time of Strategy 2 is much shorter than that of Strategy 1 as shown Fig. 13. The reason is that Strategy 2 prunes more objects than Strategy 1 as shown in Table III.

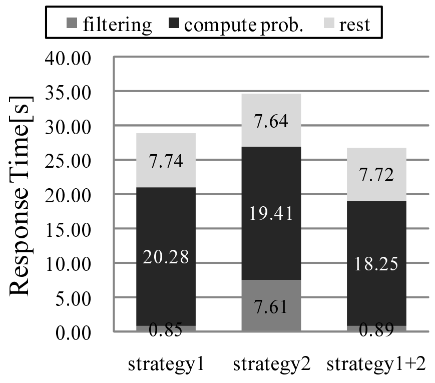

Fig. 12. Response time ($\gamma = 1$)

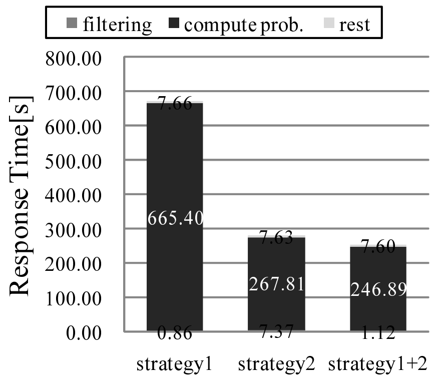

Fig. 13. Response time ($\gamma = 50$)

TABLE II
NUMBER OF CANDIDATES ($\gamma = 1$)

Str1	Str2	Str1+2
24	32	23

TABLE III
NUMBER OF CANDIDATES ($\gamma = 50$)

Str1	Str2	Str1+2
847	316	292

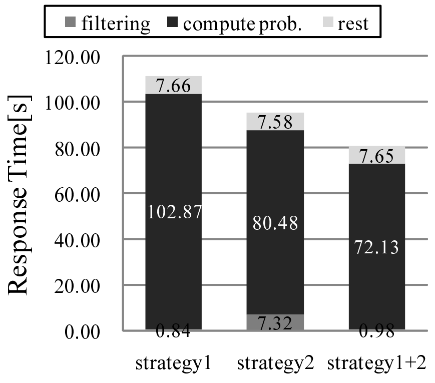

Fig. 14. Response time ($\theta = 0.03$)

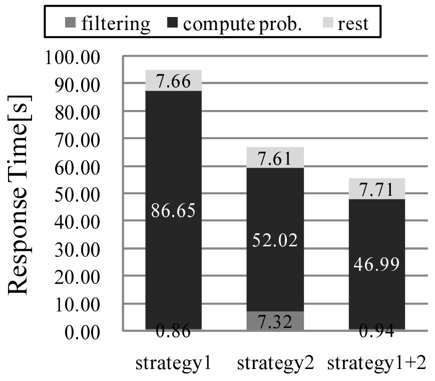

Fig. 15. Response time ($\theta = 0.05$)

TABLE IV
NUMBER OF CANDIDATES
($\theta = 0.03$)

Str1	Str2	Str1+2
128	93	82

TABLE V
NUMBER OF CANDIDATES
($\theta = 0.05$)

Str1	Str2	Str1+2
107	58	53

3) Using Different θ Values ($\theta = 0.03$ and $\theta = 0.05$): To examine the impact of the probability threshold on the performance of the strategies, we changed θ to $\theta = 0.03$ and $\theta = 0.05$.

Figures 14 and 15 show the query processing time for each strategy with $\theta = 0.03$ and $\theta = 0.05$, respectively. Tables IV and V show the number of candidate objects that required numerical integration. The number of resulting objects is 7 in the case of $\theta = 0.03$, and is 3 in the case of $\theta = 0.05$. The reason why the number of resulting objects decreases as the threshold becomes larger is clear from the definition of a probabilistic nearest neighbor query.

Figures 8, 14, and 15 show that the superiority of Strategy 2 over Strategy 1 becomes greater as θ becomes larger. As shown in Tables I, IV, and V, Strategy 2 prunes more objects (compared to Strategy 1) as θ becomes larger.

4) Using Different Shapes of the Isosurface of $p_q(\boldsymbol{x})$: In the default setting, the shape of the isosurface of $p_q(\boldsymbol{x})$ was an ellipse titled at 30° and the major-to-minor axis ratio is 3:1. However, we also considered different shaped isosurfaces. Specifically, we performed calculations for a circular isosurface and for a narrow ellipse titled at 30° with a major-to-minor axis ratio of 9:1. To let the shape of the isosurface of $p_q(\boldsymbol{x})$ be a circle, we defined the covariance matrix $\boldsymbol{\Sigma}$ in Eq. (1) to be

$$\boldsymbol{\Sigma} = \gamma \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

Similarly, to let the shape of the isosurface of $p_q(\boldsymbol{x})$ be a narrow ellipse titled at 30° with a major-to-minor axis ratio of 9:1, we defined $\boldsymbol{\Sigma}$ to be

$$\boldsymbol{\Sigma} = \gamma \begin{bmatrix} 61 & 20\sqrt{3} \\ 20\sqrt{3} & 21 \end{bmatrix}.$$

We used $\gamma = 30$ in the former case and $\gamma = 2$ in the latter case so that the numbers of resulting objects in these cases are approximately the same as in the default setting. Figures 16 and 17 show the query processing time for each strategy for the case of a circle and a narrow ellipse, respectively. Tables VI and VII show the number of candidate objects that required numerical integration. The number of resulting objects is 26 in the former case, and is 24 in the latter case.

Figure 16 shows that Strategy 2 is superior to Strategy 1 when the shape of the isosurface is a circle. On the other hand, Figure 17 shows that Strategy 1 is superior to Strategy 2 when the shape of the isosurface is a narrow ellipse. The reason is that Strategy 1 can prune more objects (compared to Strategy 2) as the shape of the isosurface becomes thinner as shown in Tables I, VI, and VII.

VI. CONCLUSIONS

In this paper, we have proposed nearest neighbor query processing methods for query objects with imprecise Gaussian-based locations. A standard nearest neighbor query method cannot be adapted to this context, since the query result is also probabilistic. Therefore, we defined the notion of a probabilistic nearest neighbor query, and proposed efficient query processing methods.

The simplest method is to compute the probability that each data object satisfies a query by integrating the Gaussian PDF over the Voronoi region for the object. However, since it is impossible to compute the integral of the Gaussian PDF analytically, costly numerical integration is required. Our methods reduce the computational cost by pruning the objects whose probabilities can be seen to be obviously smaller than the threshold without heavy computation. We proposed two query processing strategies based on this approach. Strategy 1 is based on the concept of a θ-region and Strategy 2 utilizes the notion of a smallest enclosing sphere (SES) and an upper-bounding function. Both approaches use tables, called U-catalogs, for the efficient query processing.

We performed experiments to compare three strategies Strategy 1, Strategy 2, and a hybrid approach with different parameter settings. We found that the superiority or inferiority

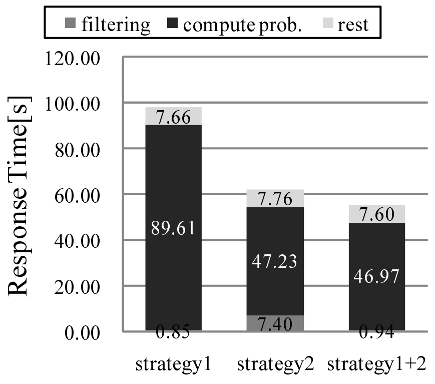

Fig. 16. Response time (circle)

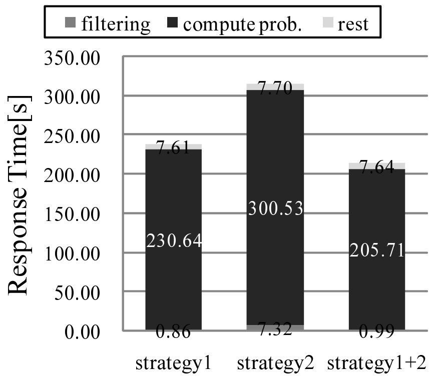

Fig. 17. Response time (narrow ellipse)

TABLE VI
NUMBER OF CANDIDATES
(CIRCLE)

Str1	Str2	Str1+2
116	53	53

TABLE VII
NUMBER OF CANDIDATES
(NARROW ELLIPSE)

Str1	Str2	Str1+2
277	383	256

of Strategies 1 and 2 depends on the given query and the specified parameters. Specifically, Strategy 1 is better than Strategy 2 when the uncertainty of the query object is smaller, the probability threshold is lower, and the shape of the isosurface is narrower. For the opposite cases, Strategy 2 shows better results. The hybrid strategy inherits the benefits of two methods and shows the best performance. For practical use, the combined strategy would be highly recommended.

In future work, we will improve the efficiency of our query processing methods using additional data structures. We are also considering the development of new methods for non-planar cases ($d = 3$, for example).

ACKNOWLEDGMENTS

This research was partly supported by a Grant-in-Aid for Scientific Research on Priority Areas, Japan (19024037).

REFERENCES

[1] “LEDA,” http://www.algorithmic-solutions.com/leda/.

[2] “RANDLIB,” http://biostatistics.mdanderson.org/SoftwareDownload/.

[3] “TIGER,” http://tiger.census.gov/.

[4] F. Aurenhammer, “Voronoi diagrams: A survey of a fundamental geometric data structure,” *ACM Computing Surveys*, vol. 23, no. 3, pp. 345–405, 1991.

[5] G. Beskales, M. A. Soliman, and I. F. Ilyas, “Efficient search for the top-k probable nearest neighbors in uncertain databases,” in *Proc. VLDB*, 2008, pp. 326–339.

[6] C. Böhm, A. Pryakhin, and M. Schubert, “The Gauss-tree: Efficient object identification in databases of probabilistic feature vectors,” in *Proc. ICDE*, 2006.

[7] ——, “Probabilistic ranking queries on Gaussians,” in *Proc. SSDBM*, 2006, pp. 119–128.

[8] J. Chen and R. Cheng, “Efficient evaluation of imprecise location-dependent queries,” in *Proc. ICDE*, 2007, pp. 586–595.

[9] R. Cheng, J. Chen, M. Mokbel, and C.-Y. Chow, “Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data,” in *Proc. ICDE*, 2008, pp. 973–982.

[10] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, “Evaluating probabilistic queries over imprecise data,” in *Proc. ACM SIGMOD*, 2003, pp. 98–109.

[11] ——, “Querying imprecise data in moving object environments,” *IEEE TKDE*, vol. 16, no. 9, pp. 1112–1127, 2004.

[12] R. Cheng and S. Prabhakar, “Managing uncertainty in sensor databases,” *SIGMOD Record*, vol. 32, no. 4, pp. 41–46, 2003.

[13] R. Cheng, S. Singh, S. Prabhakar, R. Shah, J. S. Vitter, and Y. Xia, “Efficient join processing over uncertain data,” in *Proc. CIKM*, 2006, pp. 738–747.

[14] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, “Efficient indexing methods for probabilistic threshold queries over uncertain data,” in *Proc. VLDB*, 2004, pp. 876–887.

[15] N. Dalvi and D. Suciu, “Management of probabilistic data: foundations and challenges,” in *Proc. ACM PODS*, 2007.

[16] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, 2nd ed. Springer, 2000.

[17] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Wiley, 2000.

[18] Y. Ishikawa, Y. Iijima, and J. X. Yu, “Spatial range querying for gaussian-based imprecise query objects,” in *Proc. ICDE*, 2009 (to appear).

[19] H.-P. Kriegel, P. Kunath, M. Pfeifle, and M. Renz, “Probabilistic similarity join on uncertain data,” in *Proc. DASFAA*, 2006, pp. 295–309.

[20] H.-P. Kriegel, P. Kunath, and M. Renz, “Probabilistic nearest-neighbor query on uncertain objects,” in *Proc. DASFAA*, 2007, pp. 337–348.

[21] V. Ljosa and A. K. Singh, “APLA: Indexing arbitrary probability distributions,” in *Proc. ICDE*, 2007, pp. 946–955.

[22] Y. Manolopoulos, A. Nanopoulos, A. N. Papadopoulos, and Y. Theodoridis, *R-Trees: Theory and Applications*. Springer, 2005.

[23] J. Pei, M. Hua, Y. Tao, and X. Lin, “Query answering techniques on uncertain and probabilistic data (tutorial),” in *Proc. SIGMOD*, 2008.

[24] D. Pfoser and C. S. Jensen, “Capturing the uncertainty of moving-object representations,” in *Proc. 6th Intl. Symp. on Advances in Spatial Databases (SSD’99)*, 1999, pp. 111–131.

[25] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipies: The Art of Scientific Computing*, 3rd ed. Cambridge University Press, 2007.

[26] P. Rigaux, M. Scholl, and A. Voisard, *Spatial Databases with Application to GIS*. Morgan Kaufmann, 2001.

[27] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar, “Indexing multi-dimensional uncertain data with arbitrary probability density functions,” in *Proc. VLDB*, 2005, pp. 922–933.

[28] Y. Tao, X. Xiao, and R. Cheng, “Range search on multidimensional uncertain data,” *ACM TODS*, vol. 32, no. 3, 2007.

[29] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.