ELSEVIER

Computers & Geosciences 33 (2007) 42–61

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

Uncertain spatial data handling: Modeling, indexing and query

Rui Li, Bir Bhanu*, Chinya Ravishankar, Michael Kurth, Jinfeng Ni

Center for Research in Intelligent Systems, University of California, Riverside, CA 92521, USA

Received 19 January 2006; received in revised form 14 April 2006; accepted 25 May 2006

Abstract

Managing and manipulating uncertainty in spatial databases are important problems for various practical applications of geographic information systems. Unlike the traditional fuzzy approaches in relational databases, in this paper a probability-based method to model and index uncertain spatial data is proposed. In this scheme, each object is represented by a *probability density* function (PDF) and a general measure is proposed for measuring similarity between the objects. To index objects, an *optimized Gaussian mixture hierarchy* (OGMH) is designed to support both certain/uncertain data and certain/uncertain queries. An uncertain R-tree is designed with two query filtering schemes, UR1 and UR2, for the special case when the query is certain. By performing a comprehensive comparison among OGMH, UR1, UR2 and a standard R-tree on US Census Bureau TIGER/Line® Southern California landmark point dataset, it is found that UR1 is the best for certain queries. As an example of uncertain query support OGMH is applied to the *Mojave Desert endangered species protection* real dataset. It is found that OGMH provides more selective, efficient and flexible search than the results provided by the existing trial and error approach for endangered species habitat search. Details of the experiments are given and discussed.

Keywords: Geographical information system; Spatial databases; Uncertainty; Probability density function; Indexing; Optimized Gaussian mixture hierarchy; R-tree

1. Introduction

Geographic information system (GIS) is a system of computer software, hardware, data, and personnel to help manipulate, analyze and present information that is tied to a spatial location. A spatial database management system is the system which organizes spatial information in GIS (Rigaus et al., 2001). In spatial databases, it is generally agreed that there are several types of error (uncertainty) which characterize the overall accuracy of final products. Uncertainty in GIS can arise from several sources. First, changes in the real world can cause information to become out of date, even if temporarily. Second, much of the data is acquired using automated image processing techniques applied to satellite images. Features extracted by image processing techniques have significant amounts of uncertainties. Unlike the traditional pattern recognition applications, these uncertainties are spatially variant. Simple relational database is no longer suitable for representing these uncertainties (Subrahmanian et al., 1997).

Not only can the data be certain or uncertain, the query can also be certain or uncertain, so there are

*Corresponding author. Tel.: +1 951 827 3954.
E-mail address: bhanu@cris.ucr.edu (B. Bhanu).

0098-3004/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cageo.2006.05.011

totally four combinations. In the following we give an example to explain these four scenarios.

If a tourist comes to Los Angeles and he/she is looking for the nearest fast food restaurant with the aid of Google™ Local. Following are the explicit four scenarios:

- *certain query* vs. *certain data:* The tourist knows his/her exact location, and the locations of all the fast food restaurants in the database are very accurate and up to date;
- *certain query* vs. *uncertain data:* The tourist knows his/her exact location, but the locations of the fast food restaurants in the database are in low spatial accuracy and some of them are no longer there;
- *uncertain query* vs. *certain data:* The tourist only knows he/she is near a park, but is not sure about the exact location; and the locations of all the fast food restaurants in the database are very accurate and up to date;
- *uncertain query* vs. *uncertain data:* The tourist only knows that he/she is near a park, but is not sure about the exact location; and the locations of the fast food restaurants in the database are in low spatial accuracy and some of them are no longer there.

There is an increasing awareness and some understanding of uncertainty sources in spatial data in the GIS domain (Bhanu et al., 2004a, b; Foote and Huebner, 1996; Hunter and Beard, 1992). Most of the existing approaches for management of probabilistic data are based on the relational model and use fuzzy set theory (Schneider, 1999; Robinson, 2003). They are useful for representing uncertainty at the symbolic level. However, in addition to the symbolic uncertainty, sensor-processing tasks involve uncertainties at both the numeric and the existence levels. Supporting these types of uncertainty in the current relational model using fuzzy logic is fundamentally difficult. So we need to construct a new database system framework that can handle uncertainties arising in spatial databases.

In this paper, first we present a probabilistic method to model the uncertain data, in which every object in a spatial database is represented by a *probability density* function (PDF). Second, we introduce a general similarity measure between the uncertain/certain data. Third, we design a new indexing structure, called *optimized Gaussian mixture hierarchy* (OGMH), based on the unsupervised clustering of the feature vector means. We also design a variant of R-tree with two query strategies: UR1 and UR2 to support the uncertain data/certain query scenario. Fourth, we apply our uncertainty model and the similarity model to the real data and compare OGMH, UR1, UR2 and standard R-tree on query precision, CPU cost and I/O cost for certain queries. This comparison shows that UR1 is the best for certain queries, followed by OGMH. For uncertain queries, UR1 and UR2 do not apply, while OGMH is found to be effective in a real-world application: *Mojave Desert endangered species protection*. In this application, OGMH improves the selectivity of endangered species habitat by 66% compared to the commonly used intersection method, and it is more flexible in providing the suitability of each location.

The rest of this paper is organized as follows. Section 2 presents the technical details of uncertainty modeling, similarity measure of the uncertain objects, index and query strategies. Section 3 provides the experimental results for the index comparison and the application on *Mojave Desert endangered species protection*. Finally, Section 4 concludes the paper.

2. Technical approach

2.1. Uncertain spatial data representation

In conventional spatial databases, objects are represented by fixed feature vectors in n dimensional feature space. However, when the query and the data are uncertain, a different representation is required. Error (uncertainty) in spatial databases encompasses both the imprecision of data and their inaccuracies. *Accuracy* defines the degree to which information on a map or in a digital database matches true or accepted values and *precision* refers to the level of measurement and exactness of description in a GIS database (Foote and Huebner, 1996). Compared with inaccuracy, imprecision in spatial location is very small so that it is negligible (Brown and Ehrlich, 1992). Therefore, in this paper, we deal with the errors from inaccuracy only. The inaccuracy could be *positional* inaccuracy for vector data and *attribute* inaccuracy for raster data. Usually it is described in the data quality report as a range around the true value (Brown and Ehrlich, 1992). In general multi-dimensional databases, each object is represented as a d-dimensional feature vector and the features are fixed numbers. When the data are uncertain, we need a different representation. In this paper, we use PDF

to represent each uncertain object. So it is a d-dimensional random variable, as given by

$$\mathbf{f}^n = \left[f_1^n, f_2^n, \cdots, f_d^n\right]^{\mathrm{T}}, \tag{1}$$

where $n = 1, \ldots, N$, N is the number of objects.

In this paper, we assume we know PDFs of each feature: f_j^n, $j = 1, \ldots, d$. Getting the PDFs is called *uncertainty modeling*, which is a part of our ongoing work.

In our system, we can handle both certain and uncertain data. So we do not give a specific name to the uncertain data. *Feature vector* is the name for both certain and uncertain data in this paper. The terminology *data entry* is also used when taking a database perspective.

2.2. Similarity measure

For fixed feature vectors, metrics like Euclidean distance, Manhattan distance, etc., are used to measure similarity. Since the uncertain objects are random variables represented by PDFs, we define the similarity as the probability that the two given random variables are the same, as given below:

$$similarity\ (\mathbf{D}, \mathbf{Q}) = Pr(|\mathbf{D} - \mathbf{Q}| < \Delta). \tag{2}$$

In this equation, **D** and **Q** are two objects. Usually **D** stands for data in the database and **Q** represents the query. Δ is a threshold describing the maximal error the system can tolerate and still regards **D** as "similar" to **Q**. **D**, **Q** and Δ are all vectors:$\mathbf{D} = [D_1, \ldots, D_d]$, $\mathbf{Q} = [Q_1, \ldots, Q_d]$ and $\Delta = [\Delta_1, \ldots, \Delta_d]$. Similarity is a number between 0 and 1. The more similar **D** and **Q** are, the larger the similarity is. Eq. (2) can be further expanded to

$$Pr(|\mathbf{D} - \mathbf{Q}| < \Delta) \equiv Pr\left[|D_1 - Q_1| < \Delta_1, |D_2 - Q_2| < \Delta_2, \cdots, |D_d - Q_d| < \Delta_d\right]. \tag{3}$$

So the similarity is defined as the probability that **D** is within the hyper-rectangle around **Q**, or vise versa. The hyper-rectangle size is decided by Δ.

As mentioned in the Introduction, both the data and the query can be certain and uncertain, so we define explicit similarity functions for them, as shown in Fig. 1. In this figure, F stands for the *cumulative distribution* function (CDF) and f for the PDF. These equations are written for continuous distributions. However, they can be modified to support discrete distribution easily. The random variables can either be independent or correlated, Gaussian or non-Gaussian. In special cases when the distributions are Gaussian, Puc in Fig. 1 can be replaced by Mahalanobis distance and Puu can be replaced by Bhattacharyya distance (Duda et al., 2000).

2.3. Uncertain spatial database system

The uncertainty handling spatial database system is shown in Fig. 2. In this system, it is assumed that

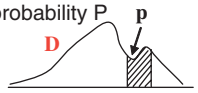
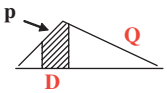
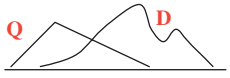
Query \ Data	Certain	Uncertain
Certain	**Physical meaning:** |D - Q| < Δ ? True / False  **Math Model:** P = Pcc(Q, D, Δ) = |D - Q| < Δ ? 1:0	**Physical meaning:** |D - Q| < Δ with probability P  **Math Model:** P = Puc(Q, D, Δ) = Pr{|D - Q| < Δ} = F_D(Q + Δ) ? F_D(Q - Δ)
Uncertain	**Physical meaning:** |D -Q| < Δ with probability P  **Math Model:** P = Puc(Q, D, Δ) = Pr{|Q - D| < Δ} =F_Q(D + Δ) - F_Q(D - Δ)	**Physical meaning:** |D - Q| < Δ with probability P  **Math Model:** P = Puu(Q, D, Δ) = ∫ Pr{ |Q - t}< Δ} · f_D(t) dt = ∫ [F_Q(t + Δ) -F_Q(t-Δ)] · f_D(t) dt

Fig. 1. Physical meaning and math model for four scenarios in our system (pictures are for 1D illustration).

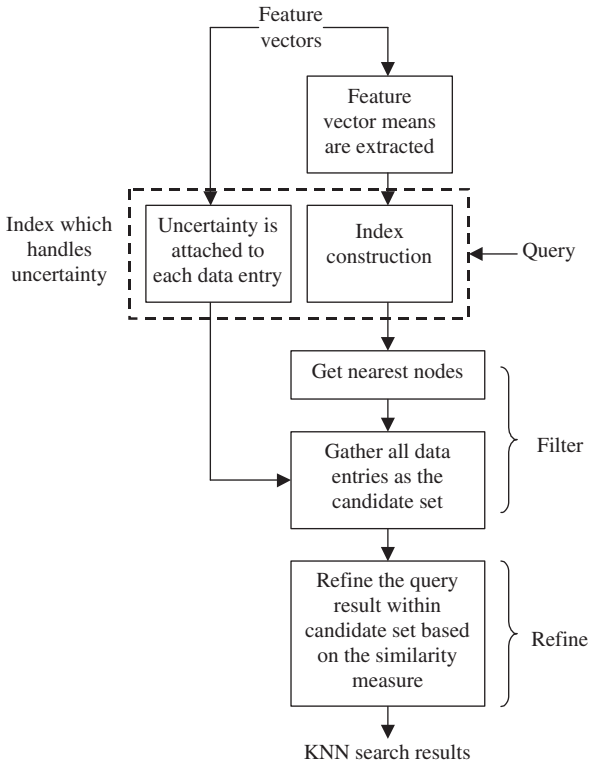


Fig. 2. System diagram for uncertainty handling spatial database.

the uncertainty is relatively small as compared to the ground truth value. Therefore, the noisy objects roughly keep the original distribution. Thus, we can use the feature vector means $\bar{\mathbf{f}}^n$, $n = 1, \ldots, N$ to construct an index, and attach uncertain information $\tilde{\mathbf{f}}^n$, $n = 1, \ldots, N$ to the corresponding data entry. This provides us an index to supports uncertain objects, as indicated by the dash-lined box in the figure.

We propose three indexing structures. The first one is called OGMH. It is based on the unsupervised clustering and it is suitable for all the four scenarios mentioned in Section 1. The other two are uncertain R-trees with two different query strategies, UR1 and UR2, which only support certain queries with uncertain data.

There are several kinds of queries in spatial databases. In this paper, we are only interested in the *K nearest neighbors* (KNN) search (e.g., find the nearest three hospitals from my house), because it is the basis for most other comprehensive queries. The KNN search is a "*filter-and-refine*" process. When a query comes in, the nearest leaf nodes are found. All the data entries (with uncertainty) belonging to these nodes are collected as the *candidate set*. This step is called the "*filter*" step. The number of nodes is decided by a predefined parameter: *minimum candidate set size* (MCS_size). It means that the total data entry number of the returned leaf nodes must not be less than the value specified by MCS_size. In the "*refine*" step, the similarity between the query and each data entry in the candidate set is calculated and the entities are sorted according to this value. The K entries, corresponding to the K largest similarities, are the search results. The "*refine*" step is the same for all the proposed index structures. The "*filter*" step differs for different indices.

The detail of the index construction and the KNN search are explained in Sections 2.3 and 2.4.

2.4. Indexing structure

2.4.1. Optimized Gaussian mixture hierarchy

The complete algorithm for OGMH construction is shown in Algorithm 1. First, the feature vectors in the dataset are classified into several subsets based on their means using an unsupervised clustering technique. These subsets give the initial leaves of the tree. Each leaf has a set of feature vectors as its data entries. Then these leaves are built into a binary tree in a bottom-up manner. Each leaf node is represented by the parameters of the Gaussian component. Each inner node is represented by the Gaussian mixture parameters of all its leaf offspring. Fig. 3 shows the parameter assignment for a four-leaf node tree.

Usually the clustering results are heavily unbalanced, which means some leaf node may have a large number of data entries, e.g., 10 times more than other leaf nodes. We define a parameter *unbalance degree* (UB_degree) as the ratio between the largest leaf size (number of data entries of a leaf) and the smallest leaf size among all the leaves. If the unbalance degree of the leaves is higher than a threshold, for all the data entries from the "large" leaf nodes, the OGMH algorithm is recursively called to generate a subtree. Then this leaf will be replaced by the subtree. This recursive procedure ends until no leaf is divisible. The clustering and tree construction are explained below in detail.

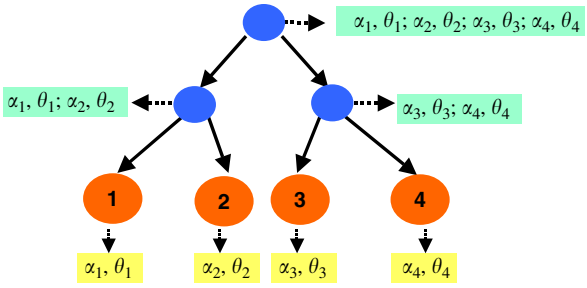


Fig. 3. Binary tree structure and parameter assignment.

Algorithm 1. Optimized Gaussian mixture hierarchy construction

Function: build_OGMH
Input: *feature vectors with uncertainty:* $\mathbf{f}^n$, $n = 1,\ldots, N$
Output: *an OGMH tree*
Begin:

1. leaves = clustering ($\bar{\mathbf{f}}^n$, $n = 1, \ldots, N$) // cluster the dataset based on the feature vector means, each leaf is a Guassian component
2. tree = tree_construction(leaves) // construct a binary tree from the leaves
3. represent each leaf node using the Gaussian component from clustering
4. **FOR** each inner node of the tree **DO**
   Assign Gaussian mixture parameters of its leaf offspring as its parameters
   END FOR
5. **FOR** $i = 1$: leaf num **DO**
   IF leaf(i) is divisible **DO**
   subtree = build_OGMH(all data of leaf(i));
   leaf(i).left_child = subtree.left_child;
   leaf(i).right_child = subtree.right_child;
   END IF
   END FOR

End

2.4.1.1. Clustering. Given a set of feature vectors, in the clustering step, we only consider their means. So it becomes a common multi-dimensional indexing problem. The means of the feature vectors in the database follow some distribution. In probability theory, any distribution can be approximated by a weighted sum of several other distributions (Duda et al., 2000), which is called *finite-mixture model*, as shown in Eq. (4). In this equation, $\mathbf{x} = [x_1, \cdots, x_d]^{\mathrm{T}}$ represents one particular outcome of a d-dimensional random variable $\mathbf{X} = [X_1, \cdots, X_d]^{\mathrm{T}}$. It is said $\mathbf{X}$ follows a C-component mixture model, where $\boldsymbol{\theta}_i$ is the set of parameters for the ith mixture component and α_i is the component weight. Then all α_i must be positive and sum up to 1. In this paper, we assume that all the components are Gaussian, so it is called *Gaussian mixture model* (GMM), where $\boldsymbol{\theta}_i$ includes the mean vector $\mathbf{u}_i$ and the covariance matrix $\boldsymbol{\Sigma}_i$.

$$p(\mathbf{X}|\boldsymbol{\theta}) = \sum_{i}^{C} \alpha_i f_i(\mathbf{X}|\boldsymbol{\theta}_i) = \sum_{i}^{C} \alpha_i N(\mathbf{u}_i, \boldsymbol{\Sigma}_i), \alpha_i > 0, \quad i = 1, \cdots, C, \text{ and } \sum_{i=1}^{C} \alpha_i = 1. \tag{4}$$

Given a set of N independent samples of $\mathbf{X}$: $X = \{\mathbf{x}^{(1)}, \cdots \mathbf{x}^{(N)}\}$, the log-likelihood corresponding to a C-component mixture is

$$\log p(X|\boldsymbol{\theta}) = \log \prod_{n=1}^{N} p(\mathbf{x}^{(n)}|\boldsymbol{\theta}) = \sum_{n=1}^{N} \log \sum_{i=1}^{C} \alpha_i p(\mathbf{x}^{(n)}|\boldsymbol{\theta}_i). \quad (5)$$

The goal is to find $\boldsymbol{\theta}$ which maximizes $\log p(X|\boldsymbol{\theta})$ (*maximum-likelihood*) or $\log p(X|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})$ (*maximum* a posteriori).

Expectation-maximization (EM) algorithm is an iterative algorithm to obtain the ML or MAP estimates of the mixture parameters (Dempster et al., 1977; McLachlan and Peel, 1997). The EM algorithm is based on the interpretation of X as incomplete data. The missing part is a set of N labels $Y = [\mathbf{y}^{(1)}, \cdots, \mathbf{y}^{(N)}]$ associated with the N samples, indicating which component produced each sample. $\mathbf{y}^{(n)} = [y_1^{(n)}, \cdots y_C^{(n)}]$. For example, $y_k^{(n)} = 1$ and $y_j^{(n)} = 0$ $(j \neq k)$ means that sample $\mathbf{y}^{(n)}$ was produced by the kth component. The complete log-likelihood is

$$\log p(X, Y|\boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{i=1}^{C} y_i^{(n)} \log\left[\alpha_i p(\mathbf{x}^{(n)}|\boldsymbol{\theta}_i)\right]. \quad (6)$$

The EM algorithm produces a sequence of estimates $\{\hat{\boldsymbol{\theta}}(t), t = 0, 1, 2, \ldots\}$ by alternatingly applying the following two steps until some convergence criterion is met:

- *E-step*: Compute the conditional expectation of the complete log-likelihood, given X and the current estimate $\hat{\boldsymbol{\theta}}(t)$. The result is the so-called R-function:

$$R(\theta, \hat{\theta}(t)) \equiv E[\log\, p(X, Y|\theta)|X, \hat{\theta}(t)] = \log\, p(X, Z|\theta). \quad (7)$$

In this equation, $Z \equiv E[Y|X, \hat{\boldsymbol{\theta}}(t)]$. Explicitly, they are given by

$$z_i^{(n)} \equiv E\left[y_i^{(n)}\middle|X, \hat{\boldsymbol{\theta}}(t)\right] = Pr\left[y_i^{(n)} = 1\middle|\mathbf{x}^{(n)}, \hat{\boldsymbol{\theta}}(t)\right] = \frac{\hat{\alpha}_i(t) p\left(\mathbf{x}^{(n)}\middle|\hat{\boldsymbol{\theta}}_i(t)\right)}{\sum_{j=1}^{C} \hat{\alpha}_j(t) p\left(\mathbf{x}^{(n)}\middle|\hat{\boldsymbol{\theta}}_j(t)\right)}. \quad (8)$$

- *M-step*: Update the parameter estimates according to Eq. (9) in the case of ML estimation or Eq. (10) for the MAP criterion

$$\hat{\boldsymbol{\theta}}(t+1) = \arg\max_{\theta} R\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}(t)\right), \quad (9)$$

$$\hat{\boldsymbol{\theta}}(t+1) = \arg\max_{\theta}\left\{R\left(\boldsymbol{\theta}, \hat{\boldsymbol{\theta}}(t)\right) + \log p(\boldsymbol{\theta})\right\}. \quad (10)$$

Figueriedo and Jain (2002) proposed a variant of EM algorithm to automatically find the number of clusters and to perform clustering. This algorithm seamlessly integrates model selection (finding the number of clusters) and model estimation (Gaussian component parameter estimation) in the iteration. It incorporates *minimum description length* (MDL) criterion for model selection and total likelihood in the L function $\mathscr{L}(\boldsymbol{\theta}, X)$ (the cost function similar to R function in Eq. (7)) and minimizes the L function given below for the best estimation of the mixture parameters.

$$\mathscr{L}(\boldsymbol{\theta}, X) = \frac{T}{2} \sum_{i:\alpha_i > 0} \log\left(\frac{N\alpha_i}{12}\right) + \frac{C_{nz}}{2} \log \frac{N}{12} + \frac{C_{nz}(T+1)}{2} - \log p(X|\boldsymbol{\theta}). \quad (11)$$

In the definition of L function in Eq. (11), T is the number of parameters specifying each component, N is the total number of samples, and C_{nz} denotes the number of non-zero-probability components. As usual, $-\log p(X|\boldsymbol{\theta})$ is the code-length of the data. The expected number of data points generated by the ith component of the mixture is $N\alpha_i$; this can be seen as an effective sample size from which $\boldsymbol{\theta}_i$ is estimated; thus the "optimal" (in the MDL sense) code length for each $\boldsymbol{\theta}_i$ is $(T/2)\log(N\alpha_i)$. The α_I's are estimated from all the N observations, giving rise to the $(C_{nz}/2)\log(N/12)$ term. This unsupervised learning process is given as Algorithm 2.

Algorithm 2. Unsupervised learning of finite mixture model algorithm

Input: $C_{\min}$, $C_{\max}$, $\varepsilon = 10^{-5}$, *initial parameters* $\hat{\boldsymbol{\theta}}(0) = \left\{\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_{C\max}, \hat{\alpha}_1, \ldots, \hat{\alpha}_{C\max}\right\}$

Output: *Mixture model in* $\hat{\boldsymbol{\theta}}_{best}$

Begin:

$t \leftarrow 0$, $C_{\mathrm{nz}} \leftarrow C_{\max}$, $L_{\min} \leftarrow +\infty$.

$u_i^{(n)} \leftarrow p\left(\mathbf{x}^{(n)}\middle|\hat{\boldsymbol{\theta}}_i\right)$, for $i = 1, \ldots, C_{\max}$, and $n = 1, \ldots, N$

WHILE $C_{nz} \geqslant C_{\min}$ **DO**

repeat

$t \leftarrow t + 1$

FOR $i = 1$ to $C_{\max}$ **DO**

$$z_i^{(n)} \leftarrow \hat{\alpha}_i u_i^{(n)} \left(\sum_{j=1}^{C_{\max}} \hat{\alpha}_j u_j^{(n)}\right)^{-1}, \text{ for } n = 1, \ldots, N$$

$$\hat{a}_i \leftarrow \max\left\{0, \left(\sum_{n=1}^{N} z_i^{(n)}\right) - \frac{T}{2}\right\} \left(\sum_{j=1}^{C\max} \max\left\{0, \left(\sum_{n=1}^{N} z_j^{(n)}\right) - \frac{T}{2}\right\}\right)^{-1}$$

$$\{\hat{\alpha}_1, \ldots, \hat{\alpha}_{\max}\} \leftarrow \{\hat{\alpha}_1, \ldots, \hat{\alpha}_{\max}\} \left(\sum_{i=1}^{C\max} \hat{\alpha}_i\right)^{-1}$$

IF $\hat{\alpha}_i > 0$, **THEN**

$\hat{\boldsymbol{\theta}}_i \leftarrow \arg\max_{\boldsymbol{\theta}_i} \log p(X, \mathscr{L}|\boldsymbol{\theta})$.

$u_i^{(n)} \leftarrow p\left(\mathbf{x}^{(n)}\middle|\hat{\boldsymbol{\theta}}_i\right)$, for $n = 1, \ldots, N$

ELSE

$C_{nz} \leftarrow C_{nz}$–1.

END IF

END FOR

$\hat{\boldsymbol{\theta}}(t) \leftarrow \left\{\hat{\boldsymbol{\theta}}_1, \ldots, \hat{\boldsymbol{\theta}}_{C\max}, \hat{\alpha}_1, \ldots, \hat{\alpha}_{C\max}\right\}$

$$\mathscr{L}[\hat{\boldsymbol{\theta}}(t), X] = \frac{T}{2} \sum_{i:\hat{\alpha}_i > 0} \log\left(\frac{N\hat{\alpha}_i}{12}\right) + \frac{C_{nz}}{2} \log\frac{N}{12} + \frac{C_{nz}(T+1)}{2} - \sum_{n=1}^{N} \log \sum_{i=1}^{C\max} \hat{\alpha}_i u_i^{(n)}$$

until $\mathscr{L}[\hat{\boldsymbol{\theta}}(t+1), X] - \mathscr{L}[\hat{\boldsymbol{\theta}}(t), X] < \mathscr{L}[\hat{\boldsymbol{\theta}}(t), X]$,

IF $\mathscr{L}[\hat{\boldsymbol{\theta}}(t), X] \leqslant \mathscr{L}_{\min}$ **THEN**

$\mathscr{L}_{\min} \leftarrow \mathscr{L}[\hat{\boldsymbol{\theta}}(t), X]$.

$\hat{\boldsymbol{\theta}}_{best} \leftarrow \hat{\boldsymbol{\theta}}(t)$

END IF

$i^* \leftarrow \arg\min_i \{\alpha_i > 0\}$, $\hat{\alpha}_{i^*} \leftarrow 0$, $C_{nz} \leftarrow C_{nz}$–1.

END WHILE

End

The iteration starts with a large component number and dynamically anneals small component and evaluates if the L function value decreases for the smaller component number at the end of each iteration. The algorithm stops when the L function value converges. We use this algorithm to get the

Gaussian components of the whole dataset. Thus, the whole dataset is divided into several groups, which form the tree leaves.

2.4.1.2. Tree construction. A binary tree is built bottom-up from the leaves obtained in the clustering step. To construct the level right above the leaves, first, a partition of leaf nodes is found so that every two nodes are merged into a new group. The *Bhattacharyya distance* (BD) (Eq. (12)) within each new group is computed. Second, all the Bhattacharyya distances for the new groups are summed up as a partition criterion, which is called the *total Bhattacharyya distance* (TBD)

$$BD:\ Bhattacharyya\_dist\,(A,B) = \frac{1}{8}(\boldsymbol{\mu}_A-\boldsymbol{\mu}_B)^{\mathrm{T}}\left[\frac{\sum_A+\sum_B}{2}\right]^{-1}(\boldsymbol{\mu}_A-\boldsymbol{\mu}_B) + \frac{1}{2}\ln\frac{|(\sum_A+\sum_B)/2|}{sqrt\,[|\sum_A||\sum_B|]}, \quad (12)$$

where random variables A and B have Gaussian distributions with means $\boldsymbol{\mu}_A$, $\boldsymbol{\mu}_B$ and covariance matrices: $\sum_A$, $\sum_B$, respectively,

$$P(A)=\frac{1}{(2\pi)^{d/2}|\sum_A|^{1/2}}\exp\left[-\frac{1}{2}(A-\boldsymbol{\mu}_A)^{\mathrm{T}}\sum_A^{-1}(A-\boldsymbol{\mu}_A)\right],$$

$$P(B)=\frac{1}{(2\pi)^{d/2}|\sum_B|^{1/2}}\exp\left[-\frac{1}{2}(B-\boldsymbol{\mu}_B)^{\mathrm{T}}\sum_B^{-1}(B-\boldsymbol{\mu}_B)\right].$$

The "best" partition is the one that minimizes the TBD. Fig. 4 is an example of the first agglomeration for a four leaf-node tree construction. There are three possible partitions for the four nodes: (1, 2|3, 4), (1, 3|2, 4) and (1, 4|2, 3), corresponding to three total Bhattacharyya distances: TBD_1, TBD_2 and TBD_3. If TBD_1 is the minimum, partition 1 is adopted. Then the next level will consist of two nodes: (1, 2) and (3, 4). In some special cases, if the component number (C in Eq. (4)) is odd or some components are far away from the others, the leftover component or the separated components are called "*isolated*" and they directly move to the next level. This agglomeration process is repeated level by level until all the nodes are merged into one group, the root. In this way, a tree is constructed.

2.4.2. Uncertain R-tree

In spatial databases, R-tree (Guttman, 1988) is the most popular indexing structure, which is a depth-balanced tree whose nodes are represented by *minimum bounding rectangles* (MBR). Fig. 5 shows an example of spatial database and Fig. 6 shows its R-tree structure. Generally, each node corresponds to a disk page. The arrow from each leaf (R8-R19) in Fig. 6 points to several data entries, for example, 10, decided by the capacity of the disk page. R-tree is designed only for the fixed data. To handle the uncertain information related to each object, it has to be modified. We construct a standard R-tree using all the feature vector means, and then attach the uncertain information of each feature vector to the corresponding data entry. It has the same tree structure as standard R-tree, except the arrows point to PDFs. The pseudo code is given in Algorithm 3.

Algorithm 3. Uncertain R-tree construction

Function build_uncertain_R-tree
Input: *uncertain feature vectors* $\mathbf{f}^n$, $n=1,\ldots,N$
Output: *Uncertain R-tree* T
Begin:

1. Get mean $\bar{\mathbf{f}}^n$ and uncertainty information $\tilde{\mathbf{f}}^n$ from each feature vector $\mathbf{f}^n$, $n=1,\ldots,N$
2. Build a classic R-tree T based on $\bar{\mathbf{f}}^n$, $n=1,\ldots,N$
3. For each data entry e in T, e.uncertainty $=\tilde{\mathbf{f}}^n$

End

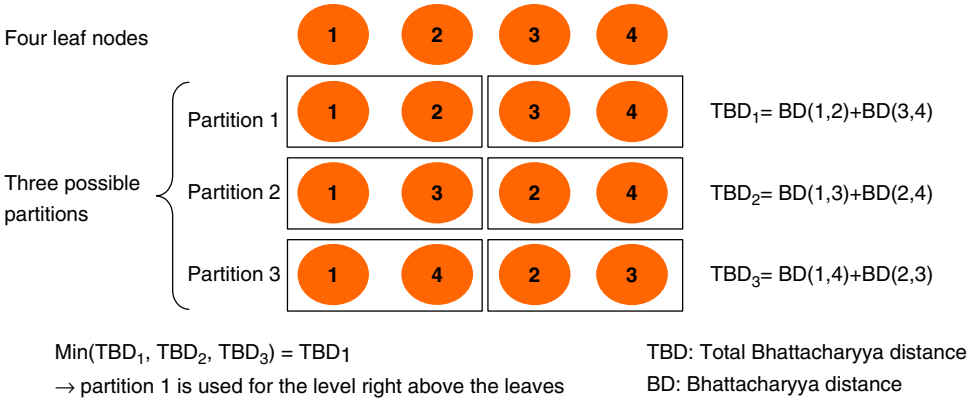


Fig. 4. Tree construction—agglomeration algorithm.

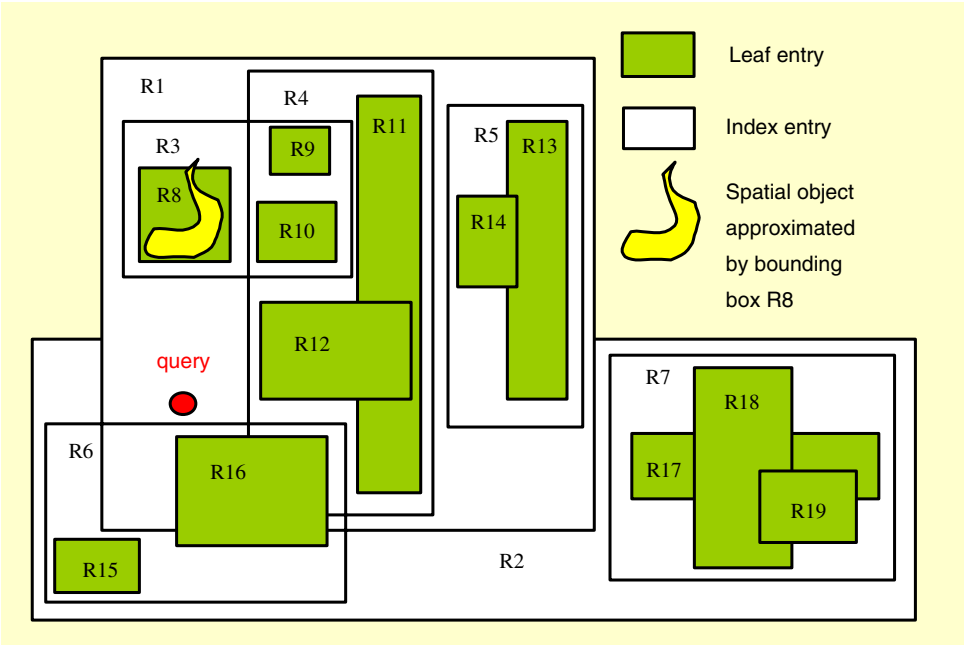


Fig. 5. Object in spatial database.

2.5. *Query—"Filter and Refine" structure for KNN search*

There are three main types of spatial queries (Ramakrishnan and Gehrke, 2000): nearest neighbor queries, spatial range queries and spatial join queries (Ni et al., 2003). In this work, we only deal with the first type; we call it the KNN search. The examples in Introduction section (four scenarios of query in uncertain databases) are KNN searches, where K is 1.

2.5.1. *KNN search for OGMH*

The KNN search algorithm for OGMH is given in Algorithm 4.

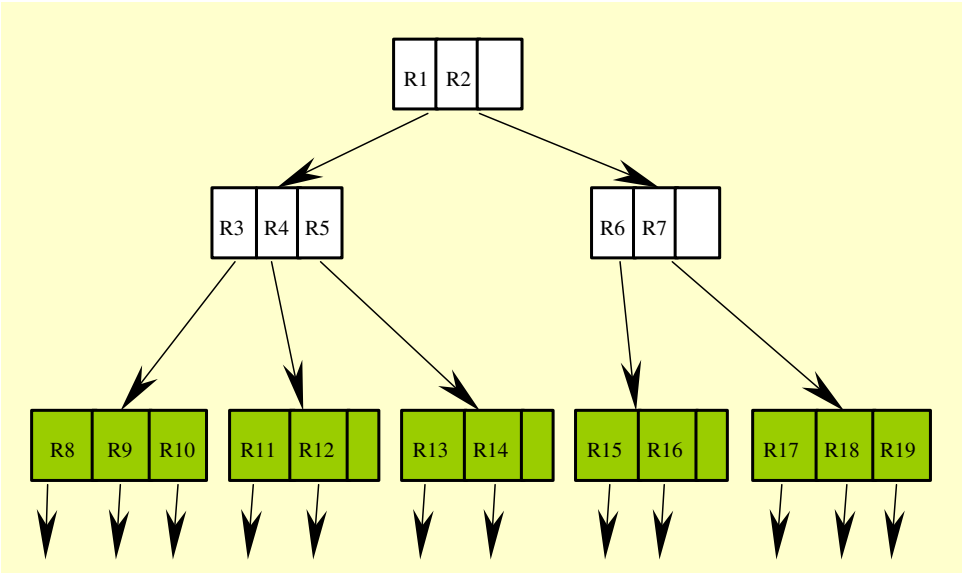


Fig. 6. R-tree structure of Fig. 5.

Algorithm 4. KNN search for OGMH

Function KNN
Input: *OGMH tree, query q*
Parameter: *MCS_size*
Output: *K nearest neighbors to q*
Begin:

1. Curr_node = OGMH.root.
2. **WHILE** curr_node is not a leaf node **DO** /****** filter step **********/
   P_L = GM_probability (curr_node→ left, *q*).
   P_R = GM_probability (curr_node→ right, *q*).
   IF P_L > P_R
   curr_node = curr_node→ left.
   ELSE
   curr_node = curr_node→right.
   END IF
   IF curr_node.size < = MCS_size
   Break
   END
   END WHILE
3. **WHILE** curr_node.total_offspring_size < MCS_size
   curr_node = curr_node→parent
   END WHILE
   Gather al the offspring data entries of curr_node as the candidate set.
4. **FOR** *i* = 1:curr_node → size **DO** /****** refinement step ******/
   $P[i]$ = curr_node → $f_i(q)$.
   END FOR
   sort(P).
5. return objects corresponding to the *K* largest elements in *P*.

End

Function GM_probability(node, q)
Input: *query q, a node in OGMH represented by its Gaussian mixture*: $\sum_i \alpha_i f_i(x)$.
Output: *the probability that q belongs to the node distribution*
Begin:
1. $p = 0$.
2. **FOR** $I = 1$: node→size **DO**
   IF q is within 3δ of node →f_i//δ: a vector consisting of standard deviations of the individual features
   $p+ =$ (node→α_i)*(node→ $f_i(q)$).
   END IF
   END FOR
3. return p.

End

When a query comes in, the tree is traversed until a leaf is reached. Depending on the value of MCS_size, other leaves (e.g., the sibling of this leaf) may also be included for refinement. All the data entries of these leaves consist of the candidate set. Then the uncertain information of each data entry is used to search for the KNN of the query.

Using mixture model to represent the inner nodes of a tree is accurate, but the calculation of Gaussian probability of the query using all the components is not efficient because the higher the level of a node, the more components it has so that calculating the similarity is more time consuming. Therefore, we calculate the similarity only when the query is within $\pm 3\delta$ (δ: a vector consisting of standard deviations of the individual features) of a mixture component. This is how the "optimized" in OGMH comes in.

2.5.2. KNN search for uncertain R-tree

To make the uncertain R-tree support KNN searches, two filter strategies: UR1 and UR2 are developed. The refinement step for them is the same.

1. *UR*1: Nearest leaves are returned, even if they do not belong to the same parents. The pseudocode is shown in Algorithm 5.

Algorithm 5. Uncertain R-tree query filter strategy 1 (UR1)

Function UR1_filter
Input: *Uncertain R-tree T, query q*
Parameter: *MCS_size*
Output: *candidate set which contains at leaset MCS_size data entries*
Begin:
1. Get the nearest t leaves in T using classic KNN R-tree query.
2. candidate_set = ϕ, candidate_set_size = 0, $i = 1$.
3. **WHILE** candidate_set_size < MCS_size
   candidate_set = candidate_set U all data entries of the ith nearest leaf.
   candidate_set_size + = size of the ith nearest leaf.
   $i = i + 1$.
   END WHILE

End

The number of leaves is decided by MCS_size. All the data entries of these leaf nodes are gathered together as the candidate set. For the query marked in Fig. 5, if 15 nearest neighbors are asked and each leaf node has 10 data entries, the nearest leaves, R12 and R16, are found. So all the data entries of R12 and R16 (20 in total) are returned as the candidate set, as shown in Fig. 7.

2. *UR*2: The nearest leaf is found, and then its ancestors are backtracked until the condition that a node with data entry offspring more than MCS_size is met. The pseudocode is described in Algorithm 6. All the data entries belonging to this node construct the candidate set. For example, for the query marked in Fig. 5, R16 is the nearest leaf. If 15 nearest neighbors are asked, R16 is backtracked until R6 is reached. So all the data entries of R6 offspring (R15 and R16) together give the candidate set, as shown in Fig. 8.

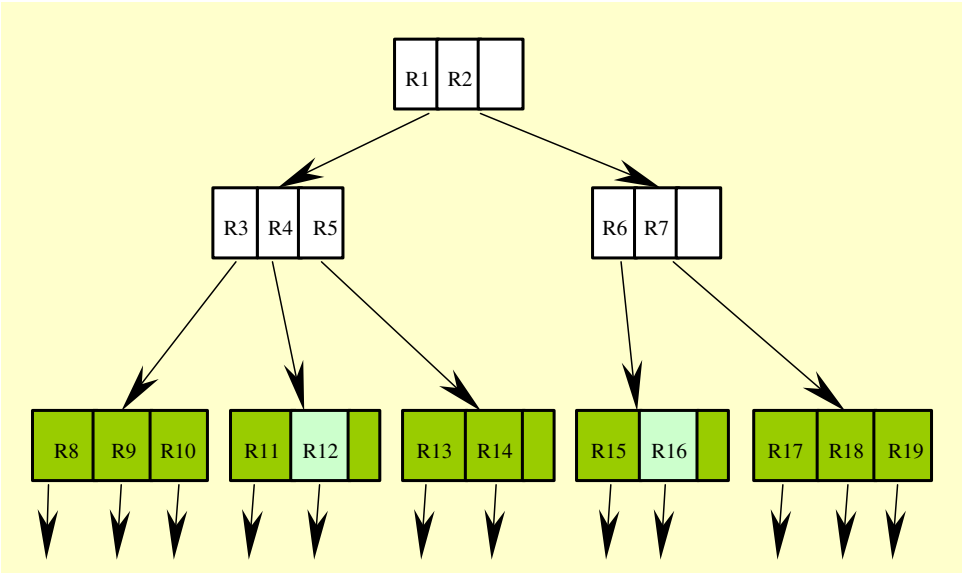


Fig. 7. Nearest neighbors of the query using UR1.

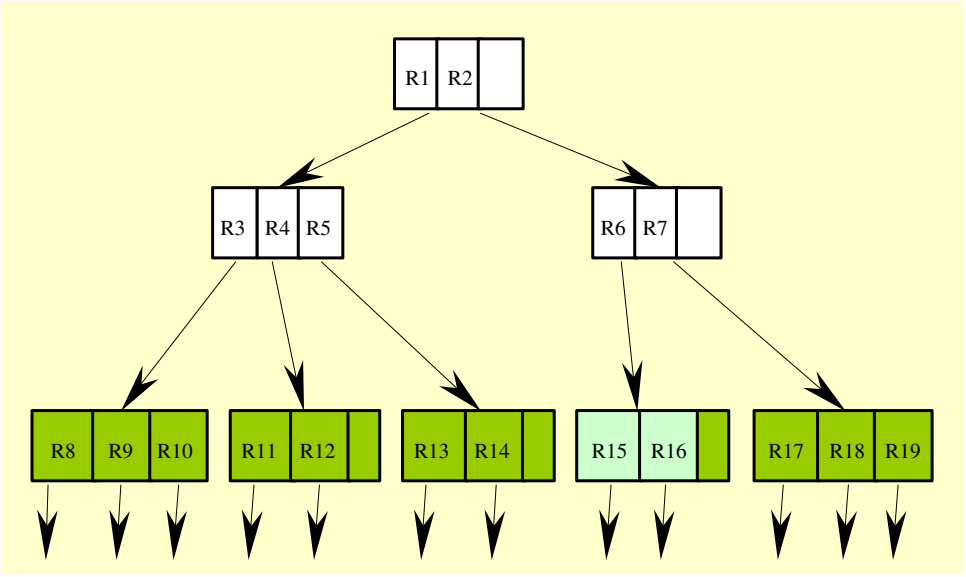


Fig. 8. Nearest neighbors of the query using UR2.

Algorithm 6. Uncertain R-tree query filter strategy 2 (UR2)

Function UR2_filter
Input: *Uncertain R-tree T, query q*
Parameter: *MCS_size*
Output: *candidate set which contains* $\geqslant$ *MCS_size data entries*
Begin:

1. Get the nearest leaf *NL* in the tree using classic R-tree query.
2. curr_node = *NL*.
3. **WHILE**
   curr_node.offspring_data_entry_num < MCS_size
   curr_node = curr_node → parent.

   END WHILE
4. Gather all the offspring data entries of curr_node as the candidate set.

End

The candidate set obtained by UR1 or UR2 is refined to get the KNN to the query. OGMH has the same filter strategy as UR2, which finds the data entry offspring of the ancestor of the nearest leaf as the candidate set. This does not guarantee the second nearest leaf is in. UR1 returns the nearest leaves as the candidate set, which will theoretically achieve equal or higher query precision. So, only the comparison between OGMH and UR2 is fair. Section 3.1 provides the details of the comparison.

3. Experimental results

In the experiments, firstly, we do comprehensive comparisons for OGMH, uncertain R-tree and standard R-tree in Section 3.1. Then we present an application of OGMH in Section 3.2.

3.1. Index comparison

3.1.1. Dataset and uncertainty assignment

We use the TIGER/Line® Southern California landmark point dataset[1] in the experiment, as shown in Fig. 9. It contains 8703 2D coordinates (x-longitude and y-latitude in degrees) and its data accuracy (uncertainty) is ± 166.67 ft (0.0005°) (Brown and Ehrlich, 1992). Uncertainty is added as a 2D Gaussian noise to each point, as shown in Eq. (13). Since measurements of longitude and latitude are independent, the noise covariance matrix is diagonal, however in general it does not have to be. Our system can support any noise, independent or correlated

$$f_{noisy} = f_{measured} + noise,$$
$$noise \sim N\left([0,0]^{\mathrm{T}}, \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}\right). \quad (13)$$

In all the experiments, the test data are fixed (using the original data) and the training data are noisy, where σ_x, σ_y for each point are randomly selected from [0, 0.0005°] or [0, 0.005°] for different uncertainty cases. 1–15 nearest neighbor(s) are returned as the result to a query.

Standard R-tree is built by following Beckmann's R*-tree structure (Beckmann et al., 1990). Uncertain R-tree is obtained from this structure.

All the programs are written in C++ and compiled by gcc 3.3.1. They run on a Sun Microsystems sun4u with 2048MB memory. The operating system is Solaris 2.8.

3.1.2. Comparison

Table 1 shows the tree parameters of the OGMH and the uncertain R-tree built from the dataset. The OGMH has fewer nodes compared with the uncertain R-tree.

The KNN search is made on OGMH, uncertain R-tree (UR1 and UR2) and standard R-tree. The experiments are performed for two different uncertainties: 0.0005°, 0.005° and two values of MCS_size: 40, 60. To minimize the effect of random initialization effect of the unsupervised mixture model learning algorithm (Figueriedo and Jain, 2002) in OGMH, in our experiments, we run the clustering/tree-building/query procedure 30 times and average the precision, I/O cost and CPU cost of each run as the overall performance.

1. *Precision comparison*: Precision is defined as the ratio between the number of correct results returned and the total number of results returned. The comparison results are shown in Figs. 10–13. From these four figures, we can make the following observations:
   (a) UR1 always gives the best performance, followed by OGMH and UR2, which is explained in Section 2.5: UR2 and OGMH have the same filter strategy, but not as good as UR1.
   (b) OGMH has higher precision than UR2, especially when MCS_size is large. So GMM is more appropriate than MBRs in object indexing.
   (c) Standard R-tree gives the worst performance precision and it is not acceptable, so in the following comparisons, standard R-tree is removed.
   From Figs. 14 and 15 we can see that when the uncertainty increases from 0.0005° to 0.005° (MCS_size = 60), the precision performance of UR1 degrades much more than

[1]The term TIGER® comes from the acronym Topologically Integrated Geographic Encoding and Referencing which is the name for the system and digital database developed at the US Census Bureau to support its mapping needs for the Decennial Census and other Bureau programs. The TIGER/Line files are a digital database of geographic features, such as roads, railroads, rivers, lakes, legal boundaries, census statistical boundaries, etc., covering the entire US. The data base contains information about these features such as their location in latitude and longitude, the name, the type of feature, address ranges for most streets, the geographic relationship to other features, and other related information. They are the public products created from the Census Bureau's TIGER database (Tiger overview, 2004. http://www.census.gov/geo/www/tiger/overview.html).

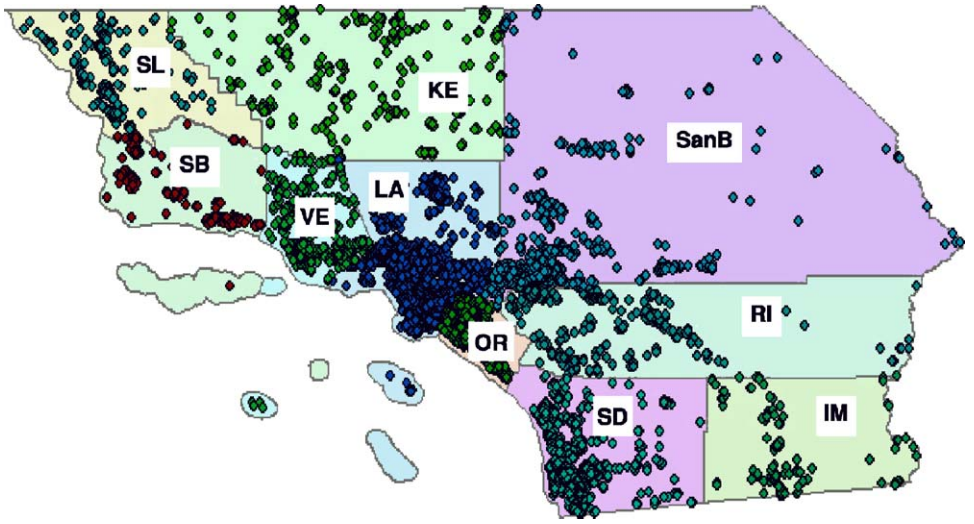


Fig. 9. Landmarks in Southern California, USA. Boundaries of various counties are labeled as: SL—San Louis, SB—Santa Barbara, VE—Ventura, LA—Los Angeles, KE—Kern, SanB—San Bernardino, RI—Riverside, IM—Imperial, SD—San Diego, OR—Orange.

Table 1
Tree parameters of OGMH, URI and UR2 on Southern CA landmark point dataset

	OGMH (average)	Uncertain R-tree
Tree node number	579	703
Tree height	17	4

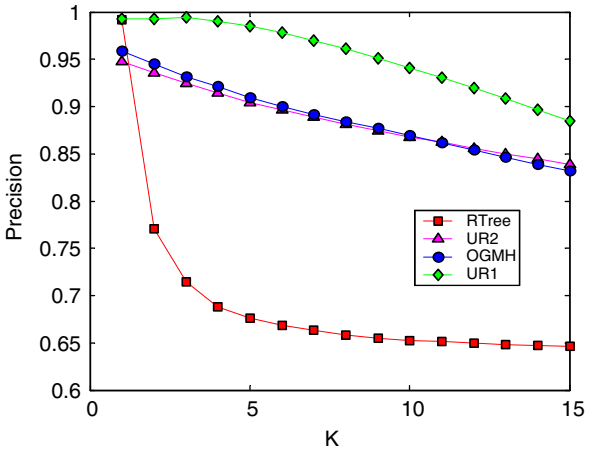


Fig. 10. Precision vs. No. of nearest neighbors K, $\sigma = 0.0005°$, MCS_size = 40.

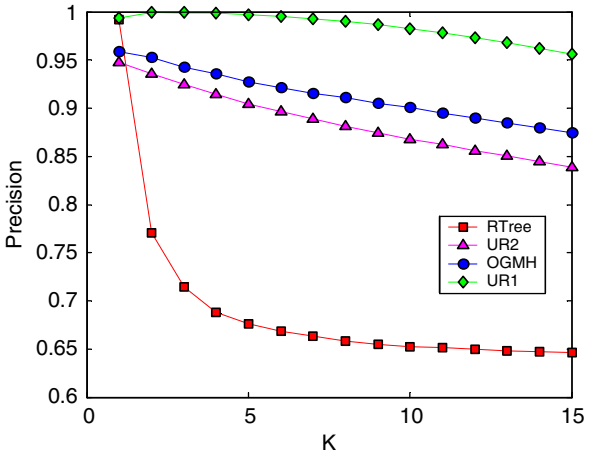


Fig. 11. Precision vs. No. of nearest neighbors K, $\sigma = 0.0005°$, MCS size = 60.

that of OGMH (0–0.5% vs. 0–0.15%). So OGMH is more stable than uncertain R-tree.

2. *I/O cost*: It is the average page access for 1-NN query. As shown in Fig. 16, UR1 needs the least I/O cost, which is approximately half of OGMH and UR2. OGMH needs more I/O because it has more levels than R-tree (17 vs. 4). The more I/O cost of UR2 comes from the back tracking. From I/O cost perspective, UR1 is the best; OGMH and UR2 are comparable.
3. *CPU cost*: It is the average time (in seconds) for 1-NN query. Fig. 17 indicates that UR1 and OGMH are comparable on time complexity and they are much more efficient than UR2, which is also due to the back tracking.

From the above three comparisons, we see that when the query is fixed, UR1 is the best on precision, I/O cost and CPU cost. OGMH has almost the same

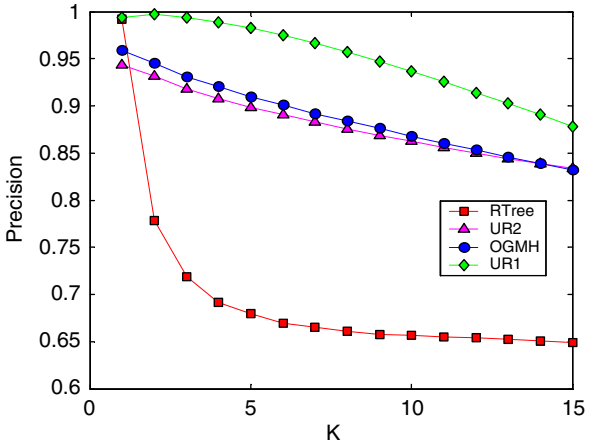


Fig. 12. Precision vs. No. of nearest neighbors K, $\sigma = 0.005°$, MCS_size = 40.

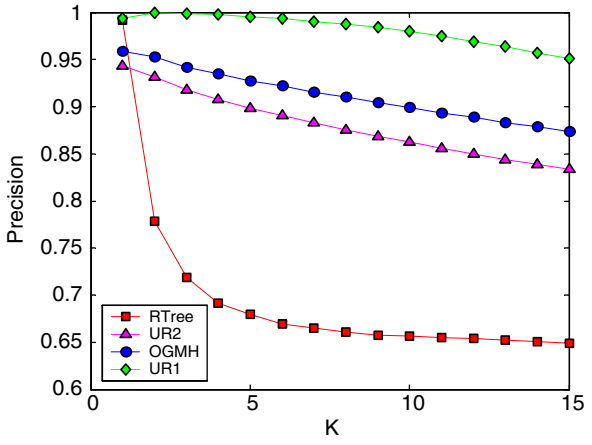


Fig. 13. Precision vs. No. of nearest neighbors K, $\sigma = 0.005°$, MCS_size = 60.

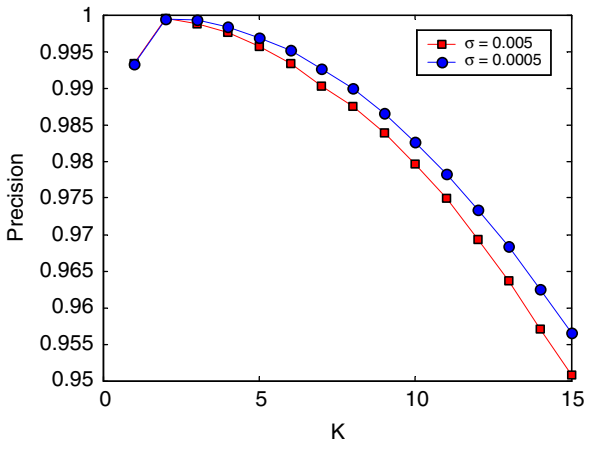


Fig. 14. UR1 precision for different σ.

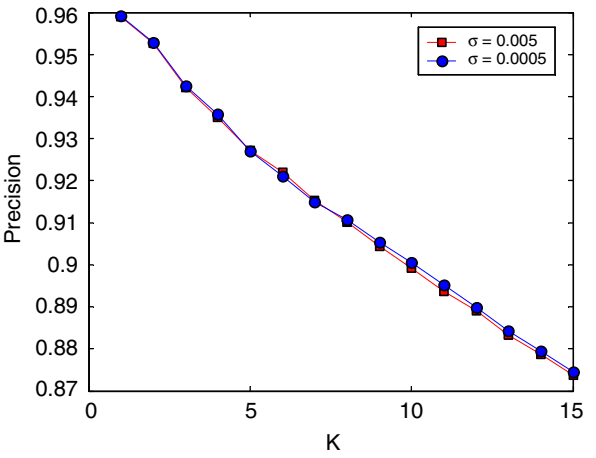


Fig. 15. OGMH precision for different σ.

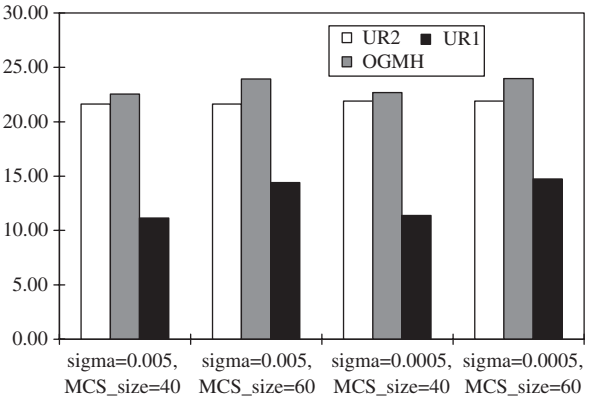


Fig. 16. I/O cost comparison among all indices.

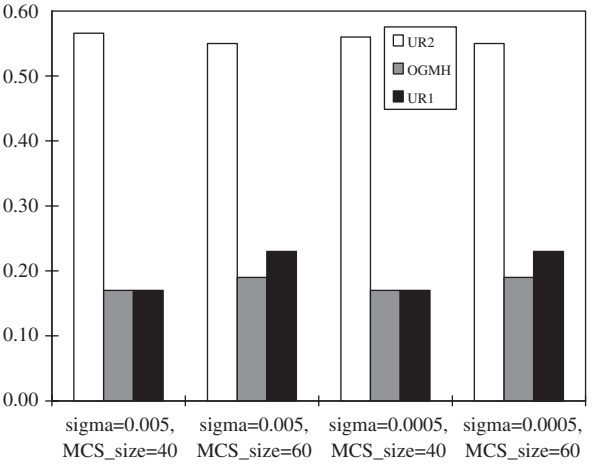


Fig. 17. CPU cost comparison among all indices.

I/O cost as UR2 but higher precision and less CPU cost. Standard R-tree is not acceptable with respect to precision performance. So OGMH is the second choice.

A complete system should be able to handle both fixed and uncertain queries. Thus, the choice of index structure and query strategy depends on the application. If no uncertain query is asked, UR1 is the best choice; otherwise only OGMH is suitable. The next section shows an application of OGMH when the query is uncertain.

3.2. An application of OGMH

In this section, we apply our uncertainty model, similarity measure and OGMH on the data from Mojave Desert endangered species (desert tortoise) protection program to show its effectiveness, efficiency and flexibility compared with the existing approach.

3.2.1. Mojave Desert species protection background

The Mojave Desert eco-region extends from eastern California to northwestern Arizona, southern Nevada, and southwestern Utah. There are hundreds of endangered species over there. Desert tortoise (*Gopherus agassizii*) is one of them. It has inhabited this region for over one million years, but its population has reduced dramatically during the last two decades due to both intrusion by people and environment change (Humphrey, 1987). It was listed as "threatened" under the California Endangered Species Act in 1989 and the federal Endangered Species Act in 1990.

Scientists have been trying to protect desert tortoises from their extinction. They are interested in finding what factors impact desert tortoises and where desert tortoises live so they can protect those places. *Mojave Desert ecosystem program* (MDEP) was the first to organize a detailed, environmentally oriented, digital geographic database set over the entire eco-region. They have made some progress on the tortoise protection. Based on the information about the tortoise habitats, MDEP biologists and researchers use ArcInfo™ to overlay and intersect the corresponding geo layers. The intersection results are places where desert tortoise might live. After this pre-processing, the area of interest reduces a lot. As the next step, they go to these areas and see if the tortoises are indeed there. If the tortoises are found, then they try to protect them. This trial and error mode is time consuming, expensive and they have to try this over and over again if they are interested in species other than tortoise.

Using our index system, we can find desert tortoise more efficiently and more flexibly. Our approach is described in Fig. 18. The idea is to partition the whole Mojave Desert into grids, describe each cell using a feature vector and index these feature vectors. Then for a given interested species, describe its habitat using a feature vector as the query and do a KNN search. Here the feature vectors in the

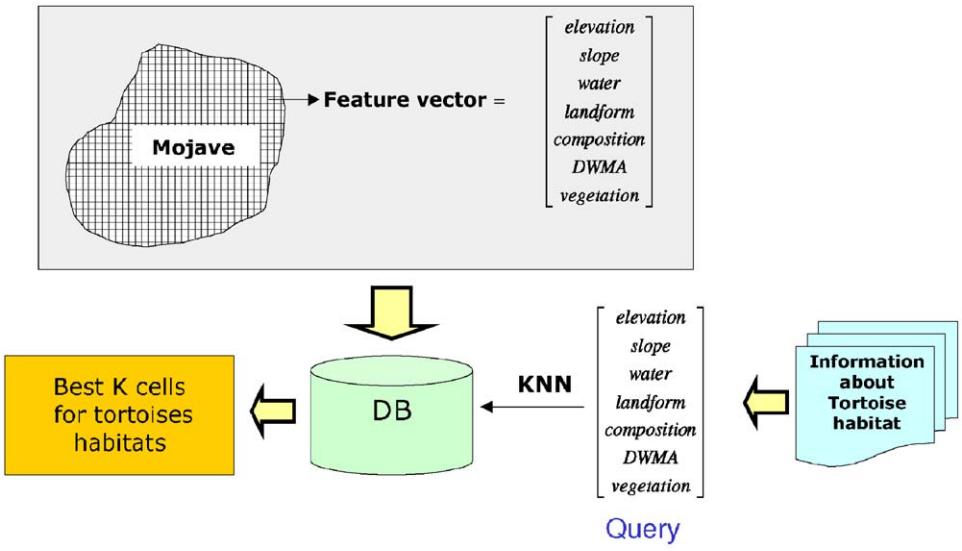


Fig. 18. Mojave Desert endangered species protection plan.

database are fixed numbers attached with a confidence and the query is uncertain: a PDF describing the geo features suitable for tortoises. The data in this database cannot be seen as *real* uncertain data because detail information is not available to construct a more sophisticated comprehensive model. But this application is an example of *certain data* vs. *uncertain query*.

3.2.2. Dataset

Our database is set up for an area of 6379.56 km^2 located at the center of Mojave Desert. It is cut into 70,884 cells. Each of them is 300 m × 300 m described by a certain feature vector and a confidence. We have selected seven geographic features based on the information about tortoise habitats (Avery et al., 1998; Boarmann and Bearman, 2002; Esque, 1993; Jennings, 1993). Table 2 shows the features we use to represent each cell. The first column is the feature type. The second column is the value range of each feature. All the feature values are normalized into the range [0, 100], as shown in the third column. Fig. 19 shows all the seven features in ArcView™.

Table 2
Features used for Mojave Desert tortoise protection

	Feature	Value	Normalized value
1	Elevation	−86 to 300 m	−4.3 to 150
2	Slope	0–71.3°	0–142.6
3	Water	0, 1, 2, 3, 4, 5	0,20,40,60,80,100
4	Landform	1,2, …, 33	3,6, …, 99
5	Composition	0, 1	0, 100
6	DWMA[a]	0, 1	0, 100
7	Vegetation	1,2, …, 34	3,6, …, 102

[a]DWMA: Desert Wildlife Management Area, which means the area that MDEP has right to access, which belongs to the National Park and Bureau of Land Management.

3.2.3. Uncertainty for both data and query

3.2.3.1. Uncertainty for data (fixed data with confidence). Original geo features are given with different resolutions. Some features are for every 30 m × 30 m cell and the others are for every 300 m × 300 m cell. So we use the 300 m × 300 m cells as our objects. This requires down sampling of the high-resolution features. Fig. 20 shows how the down sampling works. For a feature given in 30 m × 30 m cells, there are 100 such cells in a 300 m × 300 m cell. Then the feature value for this large cell is the one which has the highest occurrence among these 100 cells. Each feature of the cell is decided in this manner. The confidence of the feature vector is defined by the product of normalized occurrence of each feature.

3.2.3.2. Uncertainty for query. In this work, the query is a feature vector describing the habitat suitable for desert tortoises. From Boarmann and Bearman (2002), we summarize the PDF of each feature value shown in Table 3, which defines its probability to be a tortoise habitat. By defining these PDFs, the query is uncertain.

3.2.4. Indexing structure

Table 4 shows the total number of nodes and total number of levels of the OGMH for this dataset before and after balancing, which can give us an impression of the tree size.

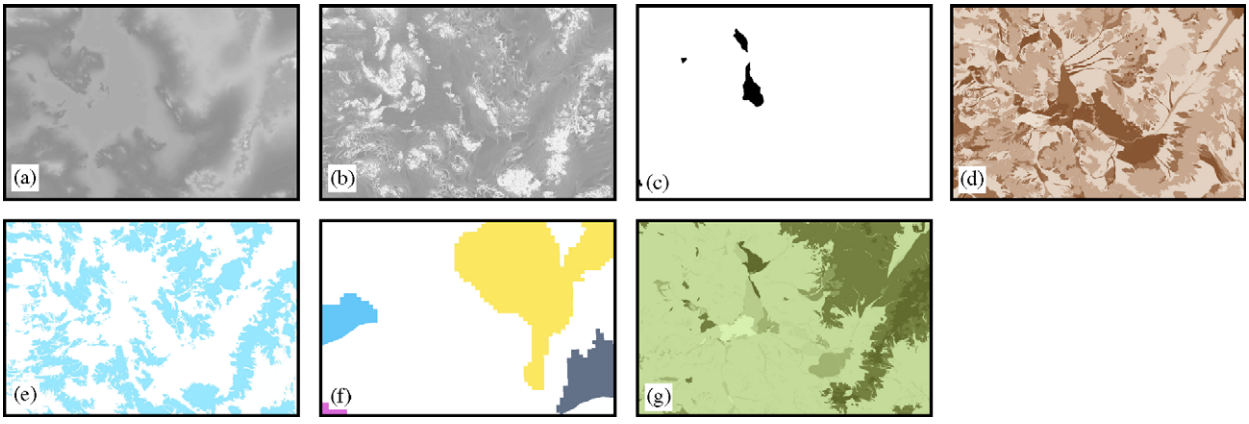


Fig. 19. Seven geo layers of Majave Desert dataset.

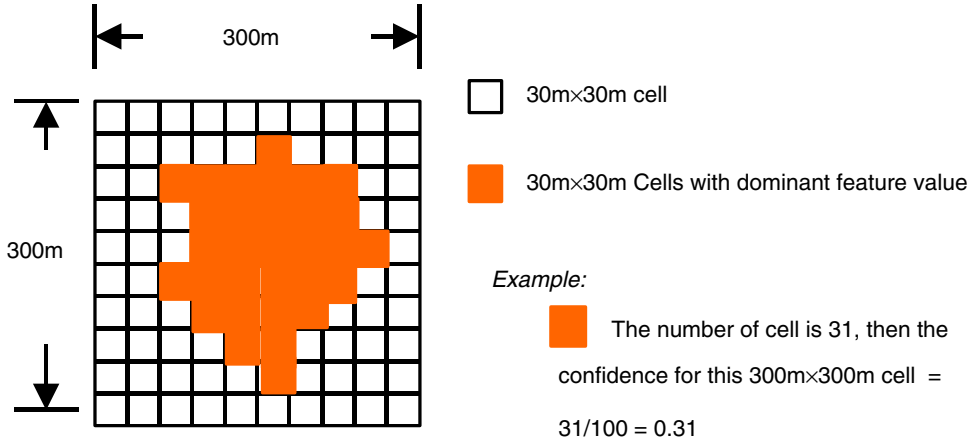


Fig. 20. Uncertainty assignment for each cell.

Table 3
PDFs for all the features

	Feature	PDF
1	Elevation	$f_1(x) = \begin{cases} 0.004785, & -4.3 < x \leqslant 150, \\ 0.004785\,e^{-0.004785(x-150)}, & 150 < x < 213.4 \end{cases}$
2	Slope	$f_2(x) = \begin{cases} 0.0176, & 0 < x \leqslant 40, \\ 0.0176\,e^{-0.0176(x-40)}, & 40 < x < 60 \end{cases}$
3	Water	$p_3(x) = \begin{cases} 0.8, & x = 0, \\ 0.1, & x = 40, 60, \\ 0, & \text{otherwise} \end{cases}$
4	Landform	$p_4(x) = \begin{cases} 0.091, & x = 3, 6, 9, 12, 21, 30, 63, 66, 81, 87, 99, \\ 0, & \text{otherwise} \end{cases}$
5	Composition	$p_5(x) = \begin{cases} 0.2, & x = 0, \\ 0.8, & x = 100 \end{cases}$
6	DWMA*	$p_6(x) = \begin{cases} 0, & x = 0, \\ 1, & x = 100 \end{cases}$
7	Vegetation	$p_7(x) = \begin{cases} 0.1429, & x = 15, 18, 39, 66, 72, 84, 99, \\ 0, & \text{otherwise} \end{cases}$

3.2.5. Results

After an OGMH is built, we make a query for the tortoise habitat. The simple intersection result based on the basic information on tortoise habitat is shown by the purple area in Fig. 21. The background is the slope, where light color means higher slope and dark color indicates lower slope.

When $\Delta = [10, 10, 20, 1, 40, 40, 1]$, for the query in Table 3, we have 8000 nearest neighbors. About 6601 of these neighbors have suitability greater than zero and they are shown by the orange areas

Table 4
OGMH parameter before and after balance

	Before balancing	After balancing
Tree node number	41	377
Tree height	8	19

Fig. 21. Intersection result from MDEP.

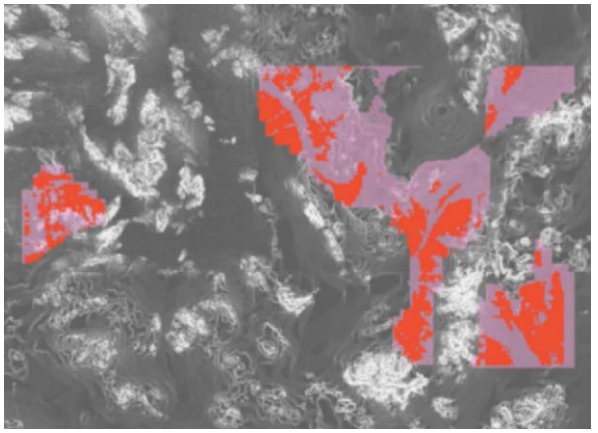


Fig. 22. Query result with suitability >0.5 (orange) when $\Delta = [10, 10, 20, 1, 40, 40, 1]$.

in Fig. 22. All these areas are suitable for living tortoises. As compared to Fig. 21, this query result provides 67.72% improvement over the intersection result shown in Fig. 21. This improvement actually reduces the extent of areas to be examined without missing out any potential tortoise habitat areas. The suitability in Fig. 22 extends from 0.5 to 1. When the user is interested in the areas with probability greater than 0.5, all these areas meet the requirement. When the user asks for areas with suitability greater than 0.75, only the red areas in Fig. 23 are suitable.

When Δ changes, query results also change. When $\Delta = [10, 15, 20, 1, 40, 40, 1]$, where tolerance of slope increase from 10 to 15, red areas in Fig. 24 have suitability over 0.75. Compared with Fig. 22, we can see more areas are suitable when Δ is higher.

From Fig. 22, we find tortoises do not like high slope. This is consistent with the information about tortoise habitats (see PDF of slope in Table 3), and further, when lower Δ is set, the results are more selective. This makes our query more flexible.

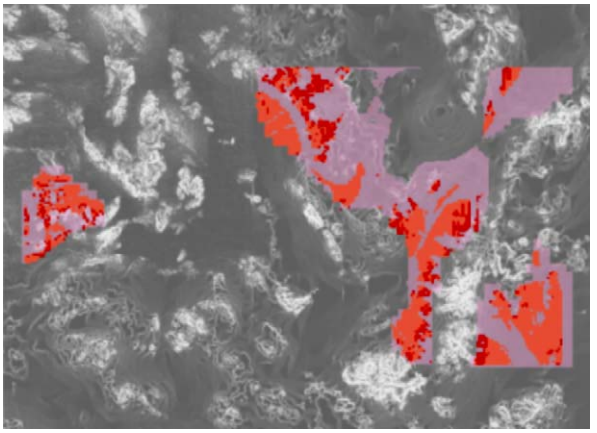


Fig. 23. Query result with suitability >0.75 (red) and <0.75 (orange) when $\Delta = [10, 10, 20, 1, 40, 40, 1]$.

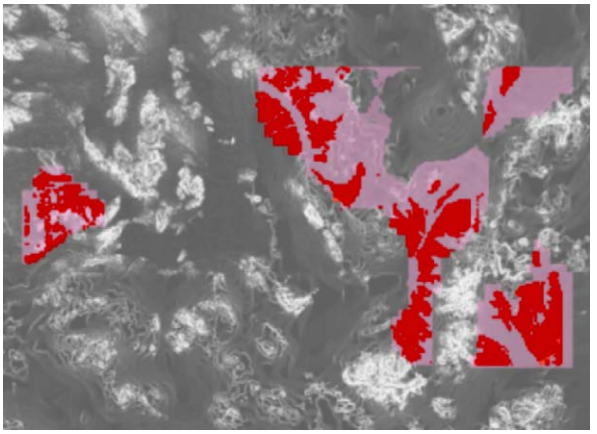


Fig. 24. Query result with suitability >0.75 (red) when $\Delta = [10, 15, 20, 1, 40, 40, 1]$.

4. Conclusions

Uncertainty is related to the data quality and decision making. In this paper, we presented a way to represent, index and query uncertain spatial data. We represented uncertain objects with PDFs and defined a similarity measure for these uncertain objects. We constructed a new *OGMH* based on GMM and uncertain R-tree with two filter strategies UR1 and UR2. Then we applied the uncertainty model, similarity measure and indexing structures on US Census Bureau TIGER/Line® dataset. After a comprehensive comparison on precision, I/O cost and CPU cost, we found that UR1 is the best for fixed queries and uncertain data. We also presented an application of *OGMH* on Mojave Desert endangered species protection database. Using our method, we found the habitats for desert tortoises and defined a confidence for each result. As compared to the result from MEDP using conventional techniques on desert tortoise, our method is more selective, more efficient and more flexible. Our method is not suitable only for the desert tortoise, but can be applied for other species. This application shows that *OGMH* is suitable for both certain/uncertain queries and certain/uncertain data.

The uncertainty modeling and executing comprehensive queries are some of the key issues that we are working on for handling uncertainty in GIS database. By adding support for uncertainty in the database management system this research will substantially increase the power and flexibility of GIS databases.

Acknowledgement

This work was supported by NSF Information Technology Research Grant 0114036. The contents of the information do not reflect the position or policy of the US Government.

References

Avery, H.W., Lovich, J.E., Neiborgs, A.G., Medica, P.A., 1998. Effects of cattle grazing on vegetation and soils in the eastern Mojave Desert, US Geological Survey, Riverside, CA.

Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B., 1990. The R*-tree: an efficient and robust access method for points and rectangles. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 322–331.

Bhanu, B., Li, R., Ravishankar, C., Kurth, M., Ni, J., 2004a. Indexing structure for handling uncertain spatial data. In: Proceedings of Sixth International Symposium on Spatial Accuracy Assessment in Natural Resources and Environmental Sciences, Portland, Maine, USA.

Bhanu, B., Li, R., Ravishankar, C.V., Ni, J., 2004b. Handling uncertain spatial data: comparisons between indexing structures. In: Proceedings of the Third International Workshop on Pattern Recognition in Remote Sensing, Kingston upon Thames, UK.

Boarmann, W.I., Bearman, K., 2002. Desert tortoises. US Geological Survey, Western Ecological Research Center, Sacramento, CA.

Brown, R.H., Ehrlich, E., 1992. TIGER/Line® Files. http://www.census.gov/geo/www/tiger/content.html.

Dempster, A., Laird, N., Rubin, D., 1977. Maximum likelihood estimation from incomplete data via the EM algorithm. Journal of the Royal Statistical Society 39, 1–38.

Duda, R.O., Hart, P.E., Stork, D.G., 2000. Pattern Classification, second ed. Wiley-Interscience Publication, New York, NY, 654pp.

Esque, T., 1993. Diet and diet selection of the desert tortoise (*Gopherus agassizzii*) in the northeast Mojave Desert. M.Sc. Thesis, Colorado State University, Fort Collins, CO, 243pp.

Figueriedo, M.A., Jain, A., 2002. Unsupervised learning of finite mixture models. IEEE Transactions on Pattern Analysis and Machine Intelligence 24, 381–396.

Foote, K.E., Huebner, D.J., 1996. Managing error, University of Texas at Austin, 〈http://www.colorado.edu/geography/gcraft/notes/manerror/manerror_f.html〉.

Guttman, A., 1988. R-Trees: A dynamic index structure for spatial searching. In: Proceedings of the ACM SIGMOD Conference, Boston, MA, pp. 47–57.

Humphrey, R., 1987. 90 Years and 535 Miles: Vegetation Changes Along the Mexican Border. University of New Mexico Press, Albuquerque, NM, 448pp.

Hunter, G.J., Beard, K., 1992. Understanding error in spatial databases. The Australian Surveyor 37, 108–119.

Jennings, W., 1993. Foraging Ecology of the Desert Tortoise (*Gopherus agassizzii*) in Western Mojave Desert. University of Texas, Arlington, TX.

McLachlan, G., Peel, D., 1997. Finite Mixture Models. Wiley-Interscience Publication, New York, NY, 456pp.

Ni, J., Ravishankar, C.V., Bhanu, B., 2003. Probabilistic spatial database operations. In: Proceedings of the Eighth International Symposium on Spatial and Temporal Databases. Lecture Notes in Computer Science. Springer, Berlin, pp. 140–158.

Ramakrishnan, R., Gehrke, J., 2000. Database Management Systems, second ed. McGraw-Hill, New York, NY, 1104pp.

Rigaus, P., Scholl, M., Voisard, A., 2001. Spatial Databases: with Application to GIS. Morgan Kaufmann, San Francisco, CA, 440pp.

Robinson, V.B., 2003. A perspective on managing uncertainty in geographic information systems with fuzzy Sets. IEEE Transactions in Geographic Information Systems 7, 211–215.

Schneider, M., 1999. Uncertainty management for spatial data in databases: fuzzy spatial data types. In: Proceedings of the Sixth International Symposium on Advances in Spatial Databases (SSD), pp. 330–351.

Subrahmanian, R.T., Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R.T., Subrahmanian, V.S., Zicari, R., 1997. Advanced Database Systems. Morgan-Kaufmann, San Francisco, CA, 576pp.