18th Int. Conf. on Scientific and Statistical Database Management (SSDBM'06)

Probabilistic Ranking Queries on Gaussians

Christian Böhm Alexey Pryakhin Matthias Schubert
Institute for Informatics
University of Munich
D-80538 Munich, Germany
{boehm,pryakhin,schubert}@dbs.ifi.lmu.de

Abstract

In many modern applications, there are no exact values available to describe the data objects. Instead, the feature values are considered to be uncertain. This uncertainty is modeled by probability distributions instead of exact feature values. A typical application of such an uncertainty model are moving objects where the exact position of each object can be determined only at discrete time intervals. Queries often involve the positions of objects between two such time stamps or after the last known time stamp. Then the objects are essentially uncertain unless the pattern of movement is very simple (e.g. linear). One of the most important probability density functions for those applications is the Gaussian or normal distribution which can be defined by a mean value and a standard deviation. In this paper, we examine a new type of queries on uncertain data objects, called probability ranking queries (PRQ). A PRQ retrieves those k objects which have the highest probability of being located inside a given query area. To speed up probabilistic queries on large sets of uncertain data objects described by Gaussians, we introduce a novel index structure called Gauss-tree. Furthermore, we provide an algorithm for employing the Gauss-tree to answer PRQs. In our experimental evaluation, we demonstrate that the Gauss-tree achieves a considerable efficiency advantage with respect to PRQs compared to other applicable methods.

1 Introduction

Recently, the research community is spending increasing effort to the development of databases that are capable to handle uncertain data objects. For an uncertain data object, some or even all describing feature values are not exactly known. However, it is possible to make conclusions about uncertain data objects by describing the uncertain values using probability distributions. For real valued features, a density distribution function can be assumed which can be used to approximate the exact feature value based on the last observed exact value. Example applications for uncertain objects are databases managing moving objects where it is not possible to determine the exact positions of objects at each point of time. Instead, the position of each observed moving object is recorded regularly after a certain time interval has passed. If a query occurs in the meantime, the exact location of the object is unknown. However, knowing the last position of an object, the current position of the object can be described by a density distribution. A similar scenario occurs in sensor networks collecting environmental data like temperature, noise level or CO_2 emissions. Usually, the sensors transfer the collected information to a database and due to bandwidth and storage limitations the exact values may be obtained in certain time intervals only. Again, if a query is posed at a time when the last exact value has been recorded some time ago, the object value becomes uncertain.

In order to manage uncertain objects in a database, an uncertainty model is needed to derive a probability distribution from the last observed feature values. A common approach which is described in [4] is to assume that there is at least a certain interval where it can be guaranteed that the current value of the data object is contained in. Within this interval an arbitrary density distribution function is specified. We will refer to this approach as the interval uncertainty model. Though there exists a large variety of probability density functions, most applications rely on standard distributions like the uniform distribution or the Gaussian distribution for each data object. A disadvantage of the interval uncertainty model is the need to specify an interval which must contain the current object value. Though it is quite often possible to make some worst-case estimation, the resulting intervals often tend to be crude approximations of the current value which might be a problem for the selectivity of query processing. A solution to this approach is the use of distribution functions like the Gaussian where it is not necessary to specify an explicit interval. Since the density of a Gaussian rapidly decreases after a given distance to the mean value is reached, the area for which it is likely that the current object value is contained in, is limited in a natural way.

In this paper, we therefore introduce another uncertainty model, called Gaussian uncertainty model. The Gaussian density distribution is one of most established ways to describe uncertainty in a variety of applications. A Gaussian is defined w.r.t. two parameters, the mean value and the standard deviation. For example, to model the change of temperature, recorded by a sensor in a sensor network, the mean value can be assumed at the last observed exact value and the variance value can be estimated based on recent variations of the observed temperatures. An important advantage of the Gaussian uncertainty model is that each object value is only complemented with one additional uncertainty attribute. Employing other distributions having p additional parameters increases the size of the database p times as well. This is a problem if we already assume limited storage capacity and bandwidth. Based on the Gaussian uncertainty model, we will discuss two important types of queries, probability threshold queries (PTQs) [5] and probability ranking queries(PRQs). The second type of queries, the PRQs, has not been studied, yet. A PRQ retrieves those k objects which have the highest probability of being located inside a given query area. To speed up processing these queries, we introduce the Gauss-tree, an index structure for efficient query processing on Gaussian density distributions. Based on the Gauss-tree, we describe algorithms for answering PRQs and PTQs. The main contributions of this paper are:

- A new model to handle uncertainty that does not rely on specifying guaranteed intervals.
- The definition of a new useful type of probabilistic queries called probability ranking queries (PRQs).
- An index structure which can be used to organize large amounts of Gaussians, called the Gauss-tree.
- Algorithms for efficiently answering PTQs and PRQs on the Gauss-tree.

The rest of this paper is organized as follows. Section 2 contains a brief description of related work in the area of indexing uncertain objects. In section 3, we define the Gaussian uncertainty model and both query types that will be discussed in the paper. The Gauss-tree and novel query algorithms are discussed in section 4. In our experimental evaluation in section 5, we demonstrate that the Gauss-tree outperforms already introduced query processing methods that are applicable to the Gaussian uncertainty model as well. Finally, section 6 concludes the paper with a summary and ideas for future work.

2 Related Work

The Gauss-tree is a member of the R-Tree family which is a spatial index structure for indexing high dimensional data. For a survey on spatial index structures please refer to [2]. The Gauss-tree was first introduced in [3] to answer so-called identification queries which are based on a Bayesian uncertainty model that cannot be used for spatial uncertainty as discussed in this paper.

In [4] a new uncertainty model is introduced and several new types of queries are described that allow the handling of inexact data. This model is based on the assumption that it is possible to determine an interval for each feature value containing the exact value. Additionally, a feature value is described by an individual probability density function over this interval. We will refer to this model as the interval uncertainty model. [5] describes two methods for efficiently answering probabilistic threshold queries that are based on the R-Tree [7]. A probabilistic threshold query returns all data objects that are placed in a given query interval with a probability exceeding a specified threshold value. The first of these methods does not rely on any assumptions about the underlying probability distributions and thus is very general. The second method is only suitable for a certain class of distribution functions, so-called symmetric and smooth variance monotonic density functions. The most prominent member of the this class of distribution functions is the Gaussian distribution. The idea of this approach is based on precalculating so-called x-bounds. An x-bound limits an area in the value set for which it can be guaranteed that any interval being completely contained within this area has a probability of less or equal to $x\%$. For storing x-bounds the method exploits the observation that the behavior of two density functions of the same type only depends on a single parameter. For Gaussians, this parameter is equivalent to the standard deviation. The differences of this approach to our new approach are the following. The method described in [5] relies on a table to approximate the properties of one type of distribution function. The Gauss-tree is for Gaussians only and thus, directly employs the Gaussian density function. In [5] a table is used to derive x-bounds for a given node in an index structure. The Gauss-tree can directly calculate the maximum probability for any Gaussian in a data node for any given query interval. Unlike the method in [5] the Gauss-tree has its own split heuristic incorporating the non-linear characteristic of the standard deviation.

[9] introduced the U-Tree for indexing uncertain 2D objects. The paper relies again on the interval uncertainty model. For the U-tree each object is guaranteed to be placed within a given polygon and a density function is given over this polygon. To index uncertain objects, the U-tree builds a conservative approximation for each node of an U-tree which consists of the minimum bounding rect-

angles (MBRs) of the polygons. The density functions are approximated by planes starting at each side of a MBR. This method is not applicable to the Gaussian uncertainty model, because the planes start on the edges of the MBR. Thus, since we do not have any guaranteed area in the Gaussian uncertainty model, the U-tree is not applicable here. Besides the mentioned methods for indexing spatially uncertain objects, [6] introduces existential uncertainty. The idea of this approach is that the existence of each data object is uncertain. Thus, each object is coupled with a probability that it is indeed real. Though this method handles uncertainty as well, the methods for query processing cannot be applied to the problems discussed in this paper.

3 Uncertainty and Query Types

3.1 Gaussian Uncertainty

An uncertain data object v is described by d uncertain attribute values v_i with $1 \leq i \leq d$. For each uncertain attribute v_i, we cannot store an exact feature value, but store a probability density function describing the likelihood of all possible attribute values. In the Gaussian uncertainty model, we consider this density distribution function to be a Gaussian which is defined as follows:

Definition 1 (Gaussian) *The Gaussian probability density function $N_{\mu,\sigma}(x)$ with respect to a mean value μ and a standard deviation σ is defined as following:*

$$N_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

The Gaussian density function is one of the most established methods to model uncertainty and is quite easy to apply. For example, in a network of temperature sensors, the last observed temperature provides a suitable mean vector and the standard deviation can be calculated from the variation of previously recorded temperatures.

To calculate the probability that an uncertain attribute value is contained in a certain query interval, we can integrate the Gaussian density function on the query interval.

Definition 2 (Gaussian Probability Function) *For $a < b$ with $a, b \in \mathbb{R}$ the Gaussian probability for a given mean value μ and a standard deviation σ can be defined as follows:*

$$P_{\mu,\sigma}(a,b) = Pr(v \in [a,b], \mu, \sigma) = \int_a^b N_{\mu,\sigma}(x)\, dx$$

An object having d uncertain attributes which are specified by a vector of mean values $\vec{\mu}$ and a vector of standard deviations $\vec{\sigma}$ is called probabilistic feature vector (pfv). For this pfv, we can calculate the probability that each attribute value v_i is contained in an attribute specific query interval $[a_i, b_i]$. Under the common assumption of attribute independency, calculating this probability can be done as follows:

$$Pr(v_i \in [a_i, b_i], \mu_i, \sigma_i, \forall i : 1 \leq i \leq d) = \prod_{i=1}^{d} P_{\mu_i,\sigma_i}(a_i, b_i)$$

3.2 Queries on the Gaussian Uncertainty Model

After describing a method to model uncertain data objects using Gaussians, we will now formally define two important types of queries on uncertain data objects. The first is the probability threshold query (PTQ) which was first defined in [4] for the interval uncertainty model. A PTQ computes all uncertain data objects that might be contained in a given query interval with a probability exceeding a given query threshold. For example, we want to retrieve all ships, that are likely to be found in a certain area of the ocean with a probability of at least 75%. Formally, a PTQ can be defined as follows:

Definition 3 (Probability Threshold Query(PTQ)) *Let DB be a set of uncertain data objects described by pfvs having d uncertain dimensions and let $t \in [0,1]$ be a probability threshold. Given d query intervals $[a_i, b_i]$ with $1 \leq i \leq d$ and $a_i, b_i \in \mathbb{R}$, a probability threshold query (PTQ) returns all objects $\vec{v} \in DB$ for which the following condition holds:*

$$Pr(v_i \in [a_i, b_i], \forall i : 1 \leq i \leq d) \geq t.$$

Let us note that if we cannot specify a query interval for one of the attributes, we may assume that the attribute is allowed to have any value. In this case, the probability for this dimension is 1 which is the integral over the complete value set of the Gaussian density function. To compute a PTQ, the straightforward approach is to retrieve each pfv in the database DB and calculate the probability that the corresponding object has attribute values which are contained in the query area. If this probability is larger than t the object is part of the result set.

PTQs are very useful in many applications. However, formulating a PTQ often proves to be more complicated then necessary. Though the given query interval might be available, finding a useful threshold probability is often difficult. Thus, a PTQ might have to be repeated with varying threshold values until a reasonable result set is found.

To avoid this problem, we introduce a new type of uncertainty queries called probability ranking queries (PRQs). A PRQ retrieves the k most likely data objects that might be placed in the given query interval. Specifying the number of results is usually much more intuitive and can easily be done by any user. In the ship example, a possible PRQ

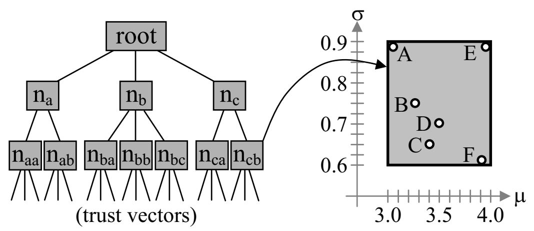


Figure 1. A 3 level Gauss-tree.

would be : "Retrieve the 10 ships which are most likely in the given area". Formally, a PRQ is defined as follows:

Definition 4 (Probability Ranking Query(PRQ)) *Let DB be a database of uncertain objects described by pfvs and let $k \in \mathbb{N}$ be a natural number. Given d query intervals $[a_i, b_i]$ with $1 \leq i \leq d, a_i < b_i, a_i, b_i \in \mathbb{R}$, a probability ranking query (PRQ) over DB returns the smallest set of data objects $kSet(\vec{a}, \vec{b})$, having at least k elements, for which the following condition holds:*

$$\forall p \in kSet(\vec{a}, \vec{b}), \forall q \in DB \setminus kSet(\vec{a}, \vec{b}) :$$
$$Pr(p_i \in [a_i, b_i], \forall i : 1 \leq i \leq d) >$$
$$Pr(q_i \in [a_i, b_i], \forall i : 1 \leq i \leq d).$$

If the number of result objects is not clear, PRQs can be extended to incremental PRQs which always retrieve the object having the next largest probability. Since the introduced query algorithms yields a close similarity to the query algorithm for nearest neighbor search described in [8], an extension to incremental queries is straight forward.

4 The Gauss-Tree

In the previous section, we have introduced the Gaussian uncertainty model and queries on top of a set of uncertain data objects. We are now going to define the Gauss-tree, a suitable index structure improving the management of uncertain object values in the Gaussian uncertainty model. Additionally, we will describe algorithms for efficiently answering PRQs and PTQs on the Gauss-tree.

4.1 The Structure of the Gauss-Tree

The Gauss-tree is a balanced tree from the R-tree family and can be used to manage d-dimensional pfvs. In contrast to the other index structures from this family, not the space of the spatial objects (i.e. the Gaussians) is indexed but instead the parameter space ($\mu_i, \sigma_i, 1 \leq i \leq d$) of the Gaussians. The structure of the index is inherited from the R-tree family which facilitates the integration into object-relational database management systems.

Definition 5 (Gauss-tree)
A Gauss-tree of degree M is a search tree where the following properties hold:

- *The root has between 1 and M entries unless it is a leaf. All other inner nodes have between $M/2$ and M entries each. A leaf node has between M and $2M$ entries.*
- *An inner node with k entries has k child nodes.*
- *Each entry of a leaf node is a probabilistic feature vector consisting of d probabilistic features (μ_i, σ_i).*
- *An entry of a non-leaf node is a minimum bounding rectangle of dimensionality $2d$ defining upper and lower bounds for every mean value $[\check{\mu_i}, \hat{\mu_i}]$ and every standard deviation $[\check{\sigma_i}, \hat{\sigma_i}]$ as well as the address of the child node.*
- *All leaf nodes are at the same level.*

In Figure 1, we see an example of a Gauss-tree consisting of 3 levels. On the right side, we have depicted the minimum bounding rectangle of a leaf node for one of the probabilistic attributes.

For query processing, we need a conservative approximation of the probability that any possible Gaussian which is stored in a node or in a certain subtree, can achieve over the given query area. In the one dimensional case, we have to compute the maximum probability of a Gaussian over the query interval $[a, b]$ under the condition that the mean value $\mu \in [\check{\mu}, \hat{\mu}]$ and the standard deviation $\sigma \in [\check{\sigma}, \hat{\sigma}]$. Since the one dimensional case can be easily extended to the multi-dimensional case by multiplying the resulting approximation probabilities, we will derive the closed form for one dimension only.

As a formula, the approximating pdf $\hat{P}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(a, b)$ is given as:

$$\hat{P}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(a, b) = \max_{\mu \in [\check{\mu},\hat{\mu}], \sigma \in [\check{\sigma},\hat{\sigma}]} \{P_{\mu,\sigma}(a, b)\}$$

For efficient query processing, a closed formula for $\hat{P}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(a, b)$ without an explicit maximization process over two continuous variables is needed. To derive this closed form, we first of all derive the following lemma.

Lemma 1 *Let $[a, b]$ with $a < b$ and $a, b \in \mathbb{R}$ be a given query interval and let $\sigma \in]0, \infty[$ be a given standard deviation. Then, the Gaussian for the given σ having the maximum probability over the interval $[a, b]$ has the mean value: $\mu_{max} = \frac{a+b}{2}$.*
Furthermore, the probability of the Gaussian decreases monotonically with the distance of μ from μ_{max}.

Proof 1 *We can differentiate $P_{\mu,\sigma}(a,b)$ by μ and see that there is only one extremum μ_{max}. Furthermore, the limes of $P_{\mu,\sigma}(a,b)$ for $\mu \to \pm\infty$ is 0. Since $P_{\mu_{max},\sigma}(a,b) > 0$, $P_{\mu,\sigma}(a,b)$ is monotonic on both sides of the maximum.*

Based on that lemma we can state that the mean value $\mu^* \in [\check{\mu}, \hat{\mu}]$ of the wanted conservative approximation is always the one closest to the middle of the query interval:

$$\mu^* = \max\{\check{\mu}, \min\{1/2(a+b), \hat{\mu}\}\}$$

To find the corresponding σ^* for the conservative approximation, we formulate the following lemma:

Lemma 2 *Let $[a,b]$ with $a < b$ and $a, b \in \mathbb{R}$ be a given query interval, let μ be a given mean value and let $[\check{\sigma}, \hat{\sigma}]$ be the interval of valid σ values with $0 < \check{\sigma} < \hat{\sigma}$. Then, we can maximize $P_{\mu,\sigma}(a,b)$ by selecting σ^* from $[\check{\sigma}, \hat{\sigma}]$ as follows:*

Case I *$a < b < \mu$:*

$$\sigma_{max} = -\frac{\sqrt{2\ln\left(\frac{\mu-b}{\mu-a}\right)(a-b)(2\mu-a-b)}}{2\ln\left(\frac{\mu-b}{\mu-a}\right)}$$

and $\sigma^* = \max\{\check{\sigma}, \min\{\sigma_{max}, \hat{\sigma}\}\}$.

Case II *$a \le \mu \le b : \sigma^* = \check{\sigma}$.*

Case III *$\mu < a < b$:*

$$\sigma_{max} = \frac{\sqrt{2\ln\left(\frac{\mu-b}{\mu-a}\right)(a-b)(2\mu-a-b)}}{2\ln\left(\frac{\mu-b}{\mu-a}\right)}$$

and $\sigma^* = \min\{\hat{\sigma}, \max\{\sigma_{max}, \check{\sigma}\}\}$.

Proof 2 ***Case I*** *We can differentiate $P_{\mu,\sigma}(a,b)$ for σ and receive the above formula for σ_{max} which is the only extremum in $]0,\infty[$. Examining the limes $\sigma \to 0$ and $\sigma \to \infty$, we observe that $P_{\mu,\sigma}(a,b)$ converges against 0 in both cases. Since $P_{\mu,\sigma_{max}}(a,b) > 0$, $P_{\mu,\sigma}(a,b)$ decreases monotonic on both sides of σ_{max}. Thus, σ^* can be chosen to be the closest value to σ_{max} in $[\check{\sigma}, \hat{\sigma}]$.*
Case II *In this case, μ is inside $[a,b]$ and if $\sigma \to 0$ then $P_{\mu,\sigma}(a,b) \to 1$. Since if $\sigma \to \infty$ then $P_{\mu,\sigma}(a,b) \to 0$ and there is no defined extremum, $P_{\mu,\sigma}(a,b)$ is monotonic and the smallest $\sigma \in [\check{\sigma}, \hat{\sigma}]$ causes the largest value for $P_{\mu,\sigma}(a,b)$.*
Case III *This case is symmetric to **case I**.*

Using both lemmas, we can calculate $P_{\mu^*,\sigma^*}(a,b)$ which is the largest possible probability for any Gaussian stored in a given node or subtree of the Gauss-tree. Let us note that this bound is tight which means that there could be indeed a Gaussian in the node having exactly the calculated probability. Figure 2 displays the probabilities for a given query interval $[a,b]$ for arbitrary μ and σ.

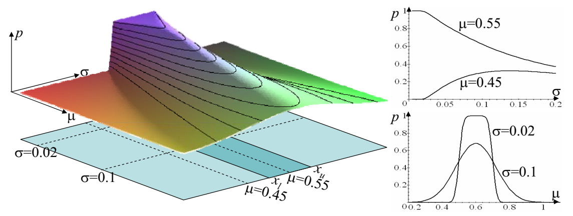


Figure 2. Visualization of probabilities for $\hat{P}_{\mu,\sigma}(a,b)$ in the μ-σ space.

4.2 Query Processing on the Gauss-Tree

After describing the structure of the Gauss-tree and deriving a conservative approximation of the maximum probability of its nodes, we are now going to describe algorithms for query processing which are suitable for answering PTQs and PRQs in efficient time.

4.2.1 PTQs

The algorithm for answering PTQs traverses the Gauss-tree from the root node in a depth-first order. Thus, the algorithm starts with inserting the subtrees of the root node into a stack. Now, the algorithms always takes the first object from the stack until the stack is empty. If the object is a node the algorithm determines μ^* and σ^* and calculates $\hat{P}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(a,b)$ for each dimension. After multiplying the probabilities for each dimension the resulting approximation is compared to the threshold t. If the approximation is smaller than t, we can prune the corresponding subtree. If not, we must push the son objects of the node onto the stack. If the object on top of the stack is a pfv, we determine its probability for lying within the query area. If this probability is larger than t we have found a result and store it for output. Let us note that this algorithm is given for demonstrating that the Gauss-tree is applicable to PTQs as well. However, the main focus of this paper are PRQs which are described in the following.

4.3 PRQs

For the answering PRQs, we employ the same idea as proposed in [8]. Instead of using a stack, the algorithm ranks the yet unprocessed entries of the Gauss-tree with a priority queue, which we will call entry queue. The entry queue has to be ordered in descending order w.r.t. to the largest probability value. Furthermore, we need a second priority queue to store the k best results being retrieved so far. This second queue is ordered in ascending order which means the result pfv having the smallest probability is al-

```
ProbabilityRankingQuery(Query $q$, integer $k$)
  $entryQueue$: ascending priority queue;
  $resultQueue$: descending priority queue;
  $entryQueue.insert(root, 1)$;
  WHILE $not entryQueue.isEmpty()$ or
    $entryQueue.getFirst() > resultQueue.getFirst()$ DO
    $currentNode = entryQueue.removeFirst()$;
    IF $currentNode$ is a data node THEN
      FOR EACH $d$ in $currentNode$ DO
        $prob =$ calculate probability of $d$ w.r.t. $q$;
        IF $resultQueue.size() < k$ THEN
          $resultQueue.insert(d, prob)$;
        ELSE IF $resultQqueue.getFirst() < prob$ THEN
          $resultQueue.removeFirst()$;
          $resultQueue.insert(d, prob)$;
        END IF
      END FOR
    ELSE IF $currentNode$ is a directory node THEN
      FOR EACH entry $e$ in $currentNode$ DO
        $prob =$ calculate probability of $e$ w.r.t. $q$;
        $entryQueue.insert(e, prob)$;
      END FOR
    END IF
  END WHILE
  RETURN $result$;
```

Figure 3. Pseudo code probability ranking query.

ways on top of the queue. We will refer to this queue as result queue.

Figure 3 denotes the algorithm in Pseudo Code. The algorithms starts with pushing the root node onto the entry queue with a probability of 1. Afterwards, we always remove the top object from the entry queue until the entry queue is empty or the algorithms can be guaranteed to have found all valid results. If the top element is a inner node, we load all son nodes, calculate their conservative approximation probabilities and insert them into the entry queue w.r.t. to these probabilities. In the case, a leaf node is placed on top of the entry queue, the exact probabilities for all pfvs stored in the node are calculated and the objects are pushed on the entry queue as well. If the top element of the entry queue is a pfv, we check if the result queue already contains k results. If not, we can add the pfv as a possible result. If we have already encountered k pfvs, we must check if the new pfv has a larger probability than the top element of the result queue. If the new pfv is a more likely result than the top of the result queue, the top of the result queue is removed and the new pfv is added to the result queue. The algorithm can be terminated if the top of the result queue has a larger probability than the top of the entry queue. In this case, it can be guaranteed that there are no pfvs which have a larger probability than the k objects in the result queue. Let us note that this algorithm is optimal since it guarantees that no unnecessary nodes are read from the hard drive.

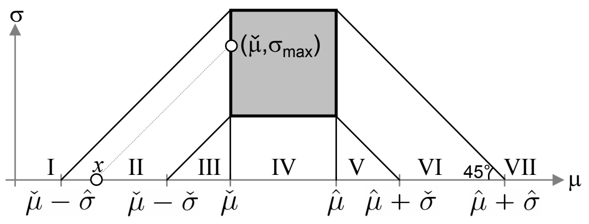

σ
$(\check{\mu}, \sigma_{max})$
I
x
II
III
IV
V
VI
45°
VII
μ
$\check{\mu} - \hat{\sigma}$
$\check{\mu} - \check{\sigma}$
$\check{\mu}$
$\hat{\mu}$
$\hat{\mu} + \check{\sigma}$
$\hat{\mu} + \hat{\sigma}$

Figure 4. The different sectors used to calculate $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$.

4.4 Tree Construction

In the following we derive the optimization goals for the insert- and split strategies applied in the Gauss-tree. Intuitively, we have to collect such probabilistic feature vectors in one common leaf node (or subtree in general) which share both similar μ and σ values because if one of these pfv is needed for a given query, also the other ones are probably needed for that query. However, the situation is not that clear as it is for conventional feature vectors where the typical optimization goal is to achieve hyper-rectangles with approximately uniform side lengths. The main difference is the following: If we have a node which contains only pfv which have a small standard deviation for one of the probabilistic features, i.e. $\hat{\sigma}_i \simeq 0$ then it is also beneficial if the μ values are spread over a small range, i.e. $\hat{\mu}_i - \check{\mu}_i \simeq 0$ because if we have both small values of σ as well as small *ranges* of μ, then this node will be very selective, i.e. the node will only be accessed for queries for which the stored pfv are highly probable candidates. In contrast, if the node also contains pfv with a high variance then a small range of μ will not help much either because the contained Gaussians will be spread over a wide range anyway. But if the range of σ values (i.e. $\hat{\sigma}_i - \check{\sigma}_i$) is small, then we know at least that this node contains no pfv with a high probability density. In this case, the node can be excluded for many queries which have already found at least k pfv with higher probability in some other nodes of the Gauss-tree. We can summarize this intuition for the split strategy (on every node overflow) in the following way: If $\hat{\sigma}_i$ is low, then perform a node split according to μ_i. Otherwise perform a split operation according to σ_i. In the following, we will capture this intuition more precisely because we do not only have to decide whether to split in μ or σ but also which of the d different μ or σ have to be used for splitting.

This mathematical model can be used not only for the decision of the split but also for resolving the situations during the insert (i.e. whenever more than one branch of the tree is eligible for the new pfv). To find a suitable criteria for the approximation quality of a node in the Gauss-tree, we first of all define the density hull for a given node $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$:

$$\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x) = \max_{\mu\in[\check{\mu},\hat{\mu}],\sigma\in[\check{\sigma},\hat{\sigma}]} \{N_{\mu,\sigma}(x)\}$$

For efficiently calculating the hull, a closed formula for $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ without an explicit maximization process over two continuous variables is needed. This can be derived by the following lemma:

Lemma 3 *The conservative approximation $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ of the probability density functions stored in a data node can be exactly computed by the following piecewise function:*

$$\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x) = \begin{cases} N_{\check{\mu},\hat{\sigma}}(x) \text{ if } x < \check{\mu}-\hat{\sigma} & (I) \\ N_{\check{\mu},\check{\mu}-x}(x) \text{ if } \check{\mu}-\hat{\sigma} \le x < \check{\mu}-\check{\sigma} & (II) \\ N_{\check{\mu},\check{\sigma}}(x) \text{ if } \check{\mu}-\check{\sigma} \le x < \check{\mu} & (III) \\ N_{x,\check{\sigma}}(x) \text{ if } \check{\mu} \le x < \hat{\mu} & (IV) \\ N_{\hat{\mu},\check{\sigma}}(x) \text{ if } \hat{\mu} \le x < \hat{\mu}+\check{\sigma} & (V) \\ N_{\hat{\mu},x-\hat{\mu}}(x) \text{ if } \hat{\mu}+\check{\sigma} \le x < \hat{\mu}+\hat{\sigma} & (VI) \\ N_{\hat{\mu},\hat{\sigma}}(x) \text{ if } \hat{\mu}+\hat{\sigma} \le x & (VII) \end{cases}$$

Proof 3 *Since $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}$ is the maximum of some other Gaussian functions $N_{\mu,\sigma}(x)$ with mean values μ between $\check{\mu}$ and $\hat{\mu}$, the hull function is monotonically increasing for all $x \le \check{\mu}$ and monotonically decreasing for all $x \ge \hat{\mu}$. Therefore, for a given x in the quadrants (I) to (III), the gaussian function which is maximal among all possible functions $N_{\mu,\sigma}(x), \mu \in [\check{\mu},\hat{\mu}], \sigma \in [\check{\sigma},\hat{\sigma}]$ must be on the left border of the minimum bounding rectangle, i.e. on the line parallel to the σ axis with $\mu = \check{\mu}$. We determine the σ value which maximizes $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}$ by setting the derivative with respect to σ to zero:*

$$\frac{\partial}{\partial\sigma} N_{\check{\mu},\sigma}(x) = 0$$

As the only positive solution we obtain a local maximum at:

$$\sigma_{max} = \check{\mu} - x$$

The function $N_{\check{\mu},\sigma}$ is also monotonically increasing with respect to σ for lower values of σ and monotonically decreasing for all $\sigma > \sigma_{max}$. For some x between $\check{\mu}-\hat{\sigma}$ and $\check{\mu}-\check{\sigma}$ our maximum is at the border of the minimum bounding rectangle, i.e. $\check{\sigma} \le \sigma_{max} \le \hat{\sigma}$, and therefore, the maximum value for some given x in quadrant (II) is

$$\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x) = N_{\check{\mu},\sigma_{max}=\check{\mu}-x}(x)$$

In quadrant (I) the local maximum is at $\sigma_{max} > \hat{\sigma}$. Due to monotonicity, the global maximum (with restriction to the minimum bounding rectangle) must be at $\hat{\sigma}$. To the same reason, the maximum is at $(\check{\mu},\check{\sigma})$ for all x in quadrant (III).

In quadrant (IV) the maximum $N_{\mu,\sigma}(x)$ is at $\mu = x$. For σ, we obtain to the same reason as for quadrant (III) a global maximum value of $\check{\sigma}$.

The cases (V) to (VII) are symmetric to (III), (II), and (I), respectively.

After we can calculate the density hull of a node, we can now use it to calculate the quality of a node. Therefore, we integrate the density hull over all possible attribute values.

$$\int_{-\infty}^{+\infty} \hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)dx$$

This integral is a good indicator for the quality of a node. If the value of the integral is 1, the quality of the node is optimal. In this case, the node contains n equal pfvs having the same μ and σ values. Therefore, we cannot make any approximation error. In general, the smaller the value of the integral the more similar is the behavior of the indexed Gaussians. Let us note that this criteria incorporates the former conclusions because the location of the mean value has less influence on the integral with increasing standard deviations. On the other hand, if the σ values of all n contained Gaussian converge against 0, then the integral will converge to the value n for n varying μ values. To conclude, the resulting nodes of a split should have an as small as possible value for this quality criteria. The integral can be determined for each probabilistic feature separately. The computation of the integral is straightforward. Remember the case analysis of lemma 3. Case (IV) is a constant function, and cases (I), (III), (V), and (VII) are Gaussian functions with given μ and σ for which efficient integration methods are known. We apply sigmoid approximation by a degree-5 polynomial. The only part which requires a little bit of consideration is case (II) and its symmetric counterpart (VI) where we have to integrate over $N_{\check{\mu},\check{\mu}-x}(x)$ from $\check{\mu}-\hat{\sigma}$ to $\check{\mu}-\check{\sigma}$. However, substituting $(\check{\mu}-x)$ for σ in the definition of the probability density function of the Gaussian distribution yields:

$$N_{\check{\mu},\check{\mu}-x}(x) = \frac{1}{\sqrt{2\pi e}\cdot(\check{\mu}-x)}$$

which integrates to $(\ln\hat{\sigma} - \ln\check{\sigma})/\sqrt{2\pi e}$ for the above mentioned integration limits.

For the insertion strategy, we apply the following rules to select a path of the Gauss-tree :

- If the new pfv fits into exactly one node, this node is followed.
- If the new vector does not fit into any node, we examine all subnodes and find the leaf node which causes the least increase of volume.
- If the new vector fits into more than one node, we follow all paths and try to find a leaf node where the node exactly fits in (or minimize the increase of volume, if no exactly fitting node exists).

When a node is beyond its capacity, it has to be split. We tentatively perform a median split in each μ-dimension and each σ-dimension of the Gauss-tree. For every tentative split, we determine the lower and upper μ and σ bounds of the two resulting nodes, and evaluate the integral $\int \hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)dx$ for both nodes. The split operation minimizing the sum of these two integrals is made permanent.

5 Evaluation

In our experimental evaluation, we implemented the Gauss-tree and its comparison partners in Java 1.4. To make the results reproducible, we measured the CPU times and counted logical page accesses on the hard drive. For calculating the complete query time, we assumed a hard drive having 6 ms access time and 50MB/s transfer rate.

We employed two data sets. The first data set (DS1) was a set of 100.000 1-dimensional Gaussians for which the μ and σ values were randomly generated. Data set 2 (DS2) was taken from the TIGER[1] database containing 2D spatial coordinates of landmarks in the US. For DS2, we used the county of Sacramento having 62.182 objects. Since we did not have any uncertainty values, we generated random standard deviations for each of the coordinates. We randomly generated 200 query intervals for DS1 or 200 query rectangles for DS2.

To have a baseline comparison partner, we compared our methods to a sequential scan over the complete database. Additionally, we implemented the method for symmetric and smooth variance-monotonic distribution functions being described in [5]. We will refer to this method as "x-bounds-tree". To extend the x-bound-tree to the multi-dimensional case, we pruned each dimension separately, i.e. when testing the pruning criteria, we assumed a maximum probability of 1 in all other dimensions. Let us note that this is not optimal, since multiplying several dimensions usually decreases the probability. However, since the method does not allow to derive a concrete maximum probability for any dimension but only checks if the closest bound is violated, this method is a feasible solution. For demonstrating the effect of our splitting and insertion method, we implemented a Gauss-tree employing the split and insertion algorithm of the x-tree [1] to which we refer to as GX-tree.

Our first experiment compares the average query time for 200 PTQs on DS1. Table 1 compares the average elapsed time for a PTQs with $t = 0.5$ and $t = 0.75$ for all 4 methods. The results indicate that all indexing techniques were capable to answer the given queries significantly faster than the sequential scan. However, all three index structures used almost exactly the same number of accessed pages for each query and used very similar CPU times. Therefore, we can conclude that the more exact approximations of the Gauss-tree do not yield an advantage when answering PTQs and the x-bounds are an efficient method for this type of queries.

threshold	Sequ. Scan	x-b. tree	GX-tree	Gauss-tree
0.5	200.3	140.0	139.0	137.6
0.75	258.1	101.4	101.6	101.1

Table 1. Comparison average query time on DS1 for PTQs.

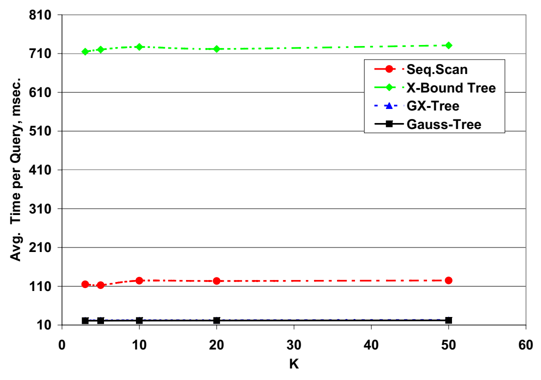
The main part of our experiments was examining the performance of the Gauss-tree when answering PRQs. To process PRQs on the x-bounds-tree, we had to find a way to rank pages w.r.t. this maximum probability. This is a problem because the described method only determines if a page can contain a pfv having a larger probability than some threshold. In order to apply ranking, we had to find a way to determine the largest probability any object in a node could have in the query interval. We solved this problem by searching the proposed ratio table for the closest x-bound to the query interval which is still outside the interval. The x corresponding to this bound was used to rank the entry queue. Let us note that the decision about pruning a node was done as proposed for PTQs in [5].

In our first experiment for PRQs, we tested all four methods for varying values of k on both data sets. The results are displayed in figure 5. The upper row of figure 5 displays the average elapsed time per query, i.e. CPU time together with calculated IO costs, and the lower row displays the observed CPU time only. As a result it can be observed that the Gauss-tree and the GX-tree retrieved the query results between 8 to 10 times faster than the sequential scan. Our adapted version of the x-bound tree worked even worse than the sequential scan w.r.t to the all over query time. However, the x-bound-tree clearly beats the sequential scan w.r.t. CPU time. Finally, the better selectivity of the Gauss-tree related methods achieves an average CPU time which again is orders of magnitudes smaller than the comparison partner on both data sets.

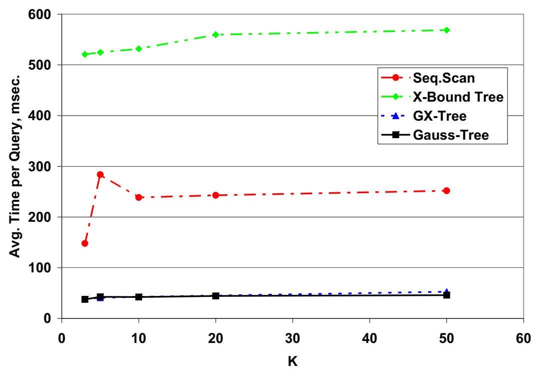
Due to the overwhelming speed up compared to the sequential scan, the figure cannot display the difference between the Gauss-tree and the GX-tree. To still demonstrate that our new split heuristic was capable to improve the structure of the tree, we display figure 6 which is a zoomed version of figure 5(a). As it can be seen the new split heuristic additional decreased the average complete query time by an additional msec..

To demonstrate that our method scales well even for larger data sets, we posed PRQs with $k = 3$ on data set DS1 and increased the size of the data base from 10.000 to 500.000. The results are displayed in figure 7. Again our adaption of the x-bound tree for PRQs did not function very

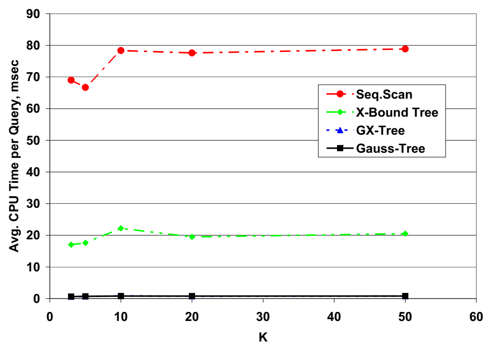
[1] available at http://www.census.gov/geo/www/tiger



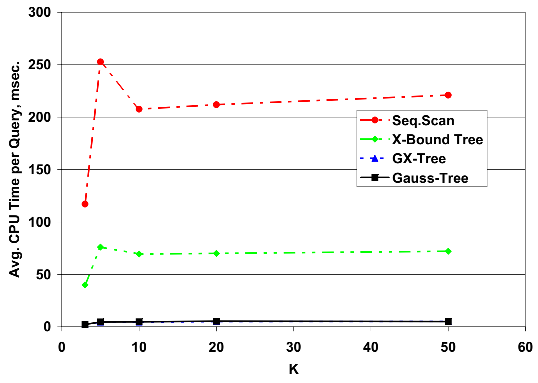
(a) average query time DS1



(b) average query time DS2



(c) average cpu time DS1



(d) average cpu time DS2

Figure 5. Complete runtime (above row) and CPU time (lower row) for PRQs for varying values of k.

well. However, the Gauss-tree and the GX-tree again display a considerable speed up which is growing with the size of the database. Thus, we can conclude that the Gauss-tree is especially well suited for very large datasets of uncertain objects modeled by Gaussians.

To conclude, the performance of the Gauss-tree for answering PTQs was rather similar to the x-bound tree in its original use. However, when answering PRQs the Gauss-tree outperformed all comparison partners by orders of magnitude. Furthermore, our novel split heuristic further improved the structure of the tree when answering PRQs.

6 Conclusions

In this paper, we introduced the Gaussian uncertainty model for describing uncertain spatial data objects. This model describes an uncertain data object as probabilistic feature vector (pfv) consisting of a mean value and a standard deviation for any uncertain feature value. Assuming a Gaussian density distribution based on these parameters, we can now determine the probability for any data object for being contained in a certain interval or (hyper-) rectangle. Applications for spatially uncertain objects are databases of sensor networks and moving objects where the exact feature value cannot be constantly monitored. To query databases of uncertain objects, we can pose probabilistic queries like probabilistic threshold queries (PTQs). A PTQ retrieves all data objects in a database that are contained in the query rectangle with a larger probability than some probability threshold t. Since the threshold is often difficult to decide, we introduced probabilistic ranking queries (PRQs) which retrieve the k data objects in a database that are contained in the query rectangle with the highest probability. To answer both types of queries in efficient time, we developed the Gauss-tree an index structure from the R-Tree family. The idea is of the Gauss-tree is to index the parameter space of the pfvs in the database. A node in the Gauss-tree contains pfvs having mean values and standard deviation being

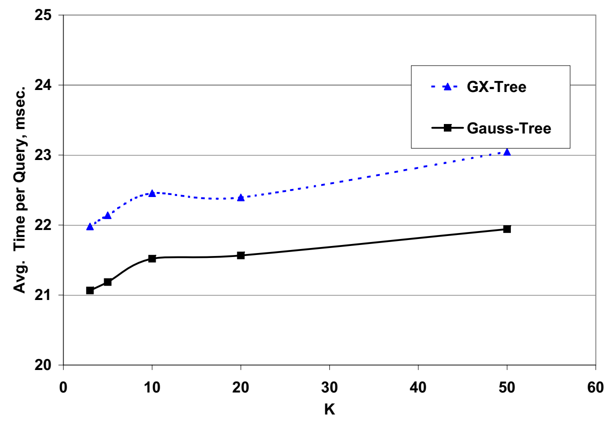


Figure 6. Average time for a PRQs for the Gauss-tree and the GX-tree.

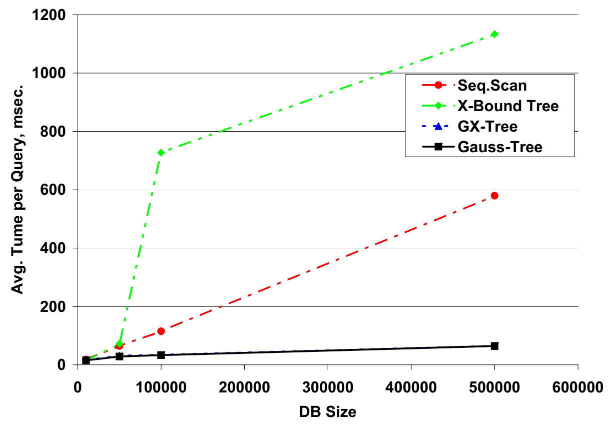


Figure 7. Average time for a PRQs for DS1 with increasing database size.

contained in a certain mean range and a certain range of standard deviations. Based on these ranges, a conservative approximation for a node and a given query rectangle can be calculated. This tight approximation is the basis of the described algorithms for answering PTQs and PRQs. The split and the insertion algorithm of the Gauss-tree is based on the density hull curve of a node. If the integral of this curve is rather small the indexed Gaussians are rather similar. Thus, the algorithm favors the splits resulting in nodes having a rather small integral over the density hull. In our experimental evaluation, we compare the Gauss-tree on both types of queries to 3 comparison partners on one artificial and two real world data sets with artificial uncertainty. The results demonstrates that the Gauss-tree achieves a query performance which is comparable to state-of-the-art methods on PTQs. For the new query type of PRQs the Gauss-tree clearly outperforms established methods which were modified to answer PRQs.

References

[1] S. Berchtold, D. A. Keim, and H.-P. Kriegel. "The X-Tree: An Index Structure for High-Dimensional Data". In *Proc. 22nd Int. Conf. on Very Large Data Bases (VLDB'96), Bombay, India*, pages 28–39, 1996.

[2] C. Böhm, S. Berchthold, and D. Keim. "Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases". *ACM Computing Surveys*, 3(33), 2001.

[3] C. Böhm, A. Pryakhin, and M. Schubert. "The Gaus-Tree: Efficient Object Identification of Probabilistic Feature Vectors". In *Proc. 22nd Int. Conf. on Data Engineering (ICDE'06)),Atlanta,GA,US*, 2006.

[4] R. Cheng, D. Kalashnikov, and S. Prabhakar. "Evaluating Probabilistic Queries over Imprecise Data". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03), San Diego, CA, USA*, pages 551–562, 2003.

[5] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. Vitter. "Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data". In *Proc. 30th Int. Conf. on Very Large Data Bases (VLDB'04), Toronto, Cananda*, pages 876–887, 2004.

[6] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis. "Probabilistic Spatial Queries on Existentially Uncertain Data". In *Proc. 9th Int. Symposium on Spatial and Temporal Databases (SSTD2005), Angra dos Reis, Brazil*, pages 400–417, 2005.

[7] A. Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching". In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'84), Boston, MA, USA*, pages 47–57, 1984.

[8] G. Hjaltason and H. Samet. "Ranking in Spatial Databases". In *Proc. 4th Int. Symposium on Large Spatial Databases, SSD'95, Portland, USA*, pages 83–95, 1995.

[9] Y. Tao, R. Cheng, X. Xiao, W. Ngai, B. Kao, and S. Prabhakar. "Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions". In *Proc. 31st Int. Conf. on Very Large Data Bases (VLDB'05), Trondheim, Norway*, pages 922–933, 2005.