22nd Int. Conf. on Data Engineering (ICDE'06), Atlanta, GA, 2006

The Gauss-Tree: Efficient Object Identification in Databases of Probabilistic Feature Vectors

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 applications of biometric databases the typical task is to identify individuals according to features which are not exactly known. Reasons for this inexactness are varying measuring techniques or environmental circumstances. Since these circumstances are not necessarily the same when determining the features for different individuals, the exactness might strongly vary between the individuals as well as between the features. To identify individuals, similarity search on feature vectors is applicable, but even the use of adaptable distance measures is not capable to handle objects having an individual level of exactness. Therefore, we develop a comprehensive probabilistic theory in which uncertain observations are modeled by probabilistic feature vectors (pfv), i.e. feature vectors where the conventional feature values are replaced by Gaussian probability distribution functions. Each feature value of each object is complemented by a variance value indicating its uncertainty. We define two types of identification queries, k-most-likely identification and threshold identification. For efficient query processing, we propose a novel index structure, the Gauss-tree. Our experimental evaluation demonstrates that pfv stored in a Gauss-tree significantly improve the result quality compared to traditional feature vectors. Additionally, we show that the Gauss-tree significantly speeds up query times compared to competitive methods.

1 Introduction

In many applications like face recognition [14, 4], fingerprint analysis [12], or voice recognition [3], data objects are represented by feature vectors with a varying degree of exactness or uncertainty. Therefore, the observed feature values cannot be considered to be known exactly and two feature vectors describing the same object can be significantly different from each other. The degree of similarity between observed and exact values can vary from feature to feature because some features cannot be determined as exactly as others. For example, it is easier to determine the proportions of a face than the breadth of a nose. Additionally, to varying uncertainties between the features, we have to consider individual uncertainties for the objects as well because the circumstances in which a given data object is transformed into a feature vector may strongly vary. For example, most data collections consisting of facial images do not just contain images that were taken under the same illumination and having exactly the same distance between camera and face.

Due to these uncertainties, we are facing new problems. An object that is observed more than once under different circumstances will most likely generate a different feature vector for each of these observations. Thus, object identification, i.e. determining if two feature vectors belong to the same object, becomes much more complicated. For example, we might have a database of facial features. When observing one of the persons that are stored in this database, we cannot simple search for the observed feature vector in the database.

To solve identification problems, the simplest solution is to employ feature based similarity search. By defining a distance function like the Euclidian distance to feature vectors, we can assume that the distance between the feature vectors corresponds to the dissimilarity of objects. Thus, to identify an object, we could retrieve the nearest neighbor in the database. To speed up query processing for large databases, a variety of index structures for feature spaces of medium to high dimensionality has been proposed, e.g. the TV-tree [10] and the X-tree [1].

However, this solution does not consider the varying uncertainties between features and between objects. Thus, the nearest neighbor might be dominated by some very uncertain feature values and the retrieved object is not the correct one. To consider varying uncertainties among each feature, the Euclidean queries could be replaced by weighted Euclidean queries or general ellipsoid queries [13]. Though, these distance measures weight the importance of each features when comparing the objects, they assume the same level of uncertainty for all database object.

To handle the uncertainty of features and objects, we propose a new model to handle inexact data in databases. This model is based on the observation that the error of measurement for a feature value is assumed to follow a normal or Gaussian distribution for most applications. Thus, we call our model the Gaussian uncertainty model. The idea of this model is to extend a feature value $\mu_{i,j}$ for data object i by an uncertainty parameter $\sigma_{i,j}$ which is corresponding to the standard deviation describing the exactness of feature j. The complete probabilistic feature vector v_i is then associated to a multivariate Gaussian distribution N_{μ_i,σ_i}. Let us note that recently the concept of uncertainty was introduced in spatial temporal databases [5, 6]. However, the introduced concepts are not applicable to identification problems. We will discuss the differences in more details in Section 2. The contributions of this paper are:

- A model to handle uncertainty in databases that is based on the assumption that the uncertainty of feature vectors can be modelled by Gaussian distributions.
- Novel types of queries called k-most-likely identification queries (k-MLIQ) and threshold identification queries (TIQ). These queries are based on the probability that a query object and a data object describe the same object.
- A general solution to calculate the probabilities that are necessary to process the introduced queries. This method can be used in combination with several data structures and query algorithms.
- An index structure for efficiently processing k-MLIQs and TIQs called the Gauss-tree. The Gauss-tree belongs structurally to the R-tree family but uses novel algorithms for query processing, insertion and tree construction.

The rest of this paper is organized as follows: Section 2 briefly surveys related work in the area of similarity search. In Section 3 the Gaussian uncertainty model is introduced and the two novel query types used in this model are defined. The algorithms to determine the exact results for k-MLIQ and TIQ are described in Section 4. These algorithms can either be used on top of a sequential scan of the complete database or be used in the refinement step for the candidate set generated by our index structure, the Gauss-tree. Section 5 defines the Gauss-tree along with the methods for query processing and tree construction. In Section 6, we give a detailed experimental evaluation of both effectiveness and efficiency of our technique and Section 7 will conclude our paper.

2 Related Work

Similarity search for high dimensional feature vectors is an important technique for information retrieval and data mining. Example applications include similarity search on structural features of 2-D contours[11], time series [7], and color histograms in image databases.To compare different feature vectors most systems employ a metric distance measure like the Euclidian distance. If some of the features are more important than others the Euclidean query can be replaced by a weighted Euclidean query or a general ellipsoid query. However, these approaches are not able to cope with individual uncertainty values for different objects. To increase the efficiency of similarity queries, various index structures have been proposed for high-dimensional feature spaces. For a survey cf. [2].

Recently, the research on probabilistic queries over uncertain data has gained increasing attention. In [5], a new uncertainty model and several new query types were proposed 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. [6] describes two methods for efficiently answering probabilistic threshold queries that are based on the R-Tree [8]. A probabilistic threshold query returns all data objects that are placed in a given query interval with a probability exceeding a specified threshold value.

Why are recent spatiotemporal uncertainty models not appropriate for identification tasks? The uncertainty model employed in [5, 6] allows to determine the probabilities that a given data object is placed in a given multidimensional interval within the query space. These probabilities are now used for a variety of queries, e.g. the already mentioned probability threshold queries. All of these queries are not directly applicable to identification tasks because the probability that two observations belong to the same data object cannot be determined by calculating the probability of containment within a certain multidimensional interval. Of course, we could assume that the query object is given by some multi-dimensional interval and retrieve the uncertain object in the database that provides the highest probability for being placed within this query interval. Besides the problem how to determine this interval for a given uncertain query, we now can apply the interval uncertainty model to object identification.

However, the resulting method has several characteristics contradicting the intuition. Consider, for instance, a query object for which all features are known with a high degree of exactness: Therefore, this object has to be associated to a very small interval. Even if we find objects in the database which fit nicely to this query, the identification probability tends to be 0 with increasing exactness of the query. Inversely, if all features of the query object are known with little certainty, this would be modelled as a large interval by conventional uncertainty models, covering almost the complete data space. Therefore, all database objects have an identification probability of 100% in this model. To conclude, the probabilities of the interval uncertainty model are not applicable to intuitively modelling identification tasks.

We will show later that it is necessary to determine the identification probability using the Bayes' theorem in order to meet the intuition that identification probabilities should be close to 1 or close to 0 for exact knowledge of both query and database object (depending on how good the actual feature values fit) and be rather indifferent (tending to $1/n$ where n is the number of objects which *could* correspond to the query object) for knowledge which is less exact.

3 The Gaussian Uncertainty Model

In this section, we formally specify inexact object representations by the concept of probabilistic feature vectors (pfv). A probabilistic feature vector v consists of d feature values μ_i and d uncertainty values σ_i where σ_i corresponds to the uncertainty of μ_i. The feature value μ_i is an observation e.g. from a sensor, and we assume that the measurement error of this sensor follows a normal distribution around the exact feature value with a known variance σ_i^2. Therefore, the data distribution of the observed values will follow a normal distribution N_{x_i,σ_i}, and the probability density that our feature value μ_i is observed, corresponds to $N_{x_i,\sigma_i}(\mu_i)$. Due to the symmetry of the Gaussians ($N_{x_i,\sigma_i}(\mu_i) \equiv N_{\mu_i,\sigma_i}(x_i)$), we can calculate $N_{\mu_i,\sigma_i}(x_i)$ to determine the probability density of the true feature value x_i for the observed feature value μ_i. This circumstance allows us to model an object by a multivariate normal distribution:

Definition 1 *A probabilistic feature vector v is a vector consisting of d pairs of feature values μ_i and standard deviations σ_i. Each pair defines a univariate Gaussian distribution of the true feature value x_i, defined by the following probability density function:*

$$N_{\mu_i,\sigma_i}(x_i) = \frac{1}{\sqrt{2\pi}\sigma_i} \cdot e^{\frac{-(x_i-\mu_i)^2}{2\sigma_i^2}}$$

The probability density of a probabilistic feature vector v for a given vector of actual values x can be calculated in the following way:

$$p(x|v) = \prod_{i=1}^{d} N_{\mu_i,\sigma_i}(x_i)$$

Our database DB consists of a set of n probabilistic feature vectors $v_i, 1 \leq i \leq n$.

3.1 Queries on a database of pfv

Deriving a probability from a density function is usually done by integration over some interval. Thus, straightforward calculation of the probability that given a pfv, we will observe some query observation q always has a probability that tends to be 0 because we would integrate over an infinitely thin interval. However, for identification tasks we can employ the fact that a given observation has to belong to one pfv from a specified set. Thus, we now can use the theorem of Bayes. This theorem allows us to calculate the conditional probability that the query q belongs to a pfv v, under the condition that q belongs to one pfv of the set of all considered pfv in DB:

$$P(v|q) = \frac{P(v) \cdot p(q|v)}{\sum_{w \in DB} (P(w) \cdot p(q|w))}$$

In this rule $p(x|v)$ is the probability density for observing x under the condition that we already observed v for the same data object. $P(v)$ ($P(w)$) is the general probability that v (w) is the answer to a query at all. In the following, we will assume that $P(v)$ ($P(w)$)is the same for any object and thus we can cancel it in the fraction. This assumption is based on the observation that it is usually not possible to anticipate the number of times that a certain object is queried.

Once we can determine this probability, we have a natural notion of how the queries for the Gaussian uncertainty model should be specified. The user can either specify a probabilistic query vector and a threshold for the probability. Then, the system has to retrieve all database objects which correspond to the query object with a probability of at least P_θ. We call this query a threshold identification query:

Definition 2 (Threshold Identification Query) ***(TIQ)*** *Let q be a probabilistic feature vector and $P_\theta \in [0 \ldots 1]$ a probability threshold. The answer of a threshold identification query is defined as follows:*

$$TIQ(q, P_\theta) = \{v \in DB | P(v|q) \geq P_\theta\}$$

An example, for a TIQ is: Give me all persons in the database that could be shown on a given image with a probability of at least 10 %.

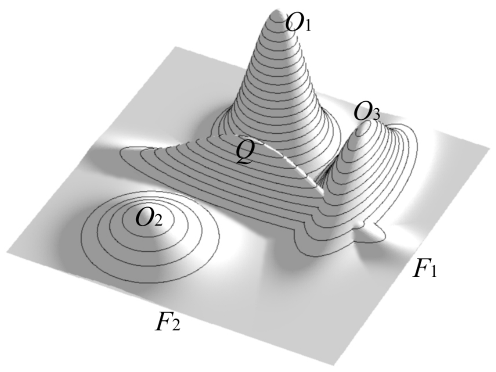

Figure 1. Probabilistic feature vectors in a 2D space. 1 query pfv and 3 database pfv.

Similarly, we can also define a k-most-likely identification query, which retrieves the k database objects providing the highest probability of belonging to the database object:

Definition 3 (k-Most-Likely Identification Query) *(k-MLIQ)* *Let DB be a database of probabilistic feature vectors v, let q be a probabilistic query vector and let $k \in \mathbb{N}$ be a natural number. Then, the answer to a k-most-likely identificaiton query (k-MLIQ) on DB is defined as the smallest set $MLIQ_k(x) \subseteq DB$ with at least k elements fulfilling the following condition:*

$$\forall v \in MLIQ_k(q), \forall w \in DB \setminus MLIQ_k(q) : \\ P(v|q) > P(w|q)$$

An example k-MLIQ is: Give the the 10 most likely persons in the database that are shown on a given image.

We will show in Section 4 how TIQ and k-MLIQ can be answered in general. This general solution is either usable as a stand-alone solution operating on top of a sequential scan of the database DB. Additionally, our general solution can also be applied as a refinement step following after a filter step (e.g. by an appropriate index structure) for efficiency improvement. Several approximation techniques can be used as filter step, e.g. approximation by intervals. However, to guarantee correctness and completeness of the result, it is necessary to define a filter which guarantees no false dismissals (false hits are removed in the following refinement step). Therefore, we propose an index structure guaranteeing no false dismissals in Section 5.

Figure 1 displays probabilistic feature vectors generated from 3 facial images of varying quality that are stored in a database and one for a query image. While feature F_1 is particularly sensitive to the rotational angle, F_2 is sensitive to illumination. The object O_1 is taken under good conditions (both features are relatively accurate), whereas for O_2 both rotation angle and illumination were bad. For O_3 the rotation was bad but the illumination was good. For the query object, in contrast, the rotation was good, but illumination was bad. We can easily recognize, that O_3 must be the object providing the highest probability for describing the same object as specified by the query. Our model derived in Section 4 will evaluate probabilities which correspond to this intuition: 77% for O_3 in contrast to 10% for O_1 and 13% for O_2. Therefore, a k-MLIQ with $k = 1$ would report O_3 as result. A TIQ with a threshold probability $P_\theta = 12\%$ would additionally report O_2.

Since conventional similarity search does not consider the individual uncertainties, a similarity query using the Euclidean distance would obtain three rather similar distances ($d(Q, O_1) = 1.53$, $d(Q, O_2) = 1.97$, $d(Q, O_3) = 1.74$). Thus in our example, the nearest neighbor would be O_1 which is excluded when considering the variances. Thus, employing ordinary feature vectors cannot be used to draw conclusions about their probabilistic feature vectors having the feature vectors as mean vectors.

4 Our General Solution

To answer any query over a database DB of probabilistic feature vectors (pfv) with respect to a probabilistic query vector q, we have to model the probability that two probability distributions given by the query pfv q and a database pfv v correspond to the same true object. This yields again a probability density function p(q—v). If the query object q would be an exact feature vector, we could calculate this probability density as mentioned in Section 3. However, if both objects are pfv, we have to consider all possible positions of the true feature vector when calculating $p(q|v)$. Then, the complete probability density corresponds to the integral over all these possible positions. Formally, we have to determine the probability density that a value x is the true feature value of both database and query object which implies the following term for each of the probabilistic features v_i and $q_i (i = 1, \ldots, d)$:

$$p(q_i|v_i) = \int_{-\infty}^{+\infty} p(v_i|x)p(q_i|x)dx$$

Remember that we are allowed to switch the mean value and the observed value due to the symmetry of the Gaussians. The term can be computed using the following lemma:

Lemma 1 (Joint probability) *Let $v_i = (\mu_v, \sigma_v)$ be a probabilistic feature of a database object and $q_i = (\mu_q, \sigma_q)$ the corresponding probabilistic feature of the query object. Then, the joint probability can be determined in the follow-*

ing way:

$$p(q_i|v_i) = \int_{-\infty}^{+\infty} N_{\mu_v,\sigma_v}(x) \cdot N_{\mu_q,\sigma_q}(x)dx = N_{\mu_v,\sigma_v+\sigma_q}(\mu_q)$$

Proof 1 *The probability density of the product*

$$N_{\mu_v,\sigma_v}(x) \cdot N_{\mu_q,\sigma_q}(x)$$

can be rewritten in the following way (this step can be proven by substitution of the definition of the normal distribution $N_{\mu,\sigma}(x) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-(\mu-x)^2/(2\sigma)}$ which is lengthy and left out due to space limitations):

$$= \frac{1}{\sqrt{2\pi}(\sigma_v + \sigma_q)} \mathrm{e}^{-\frac{(\mu_v-\mu_q)^2}{2(\sigma_v+\sigma_q)}} \cdot N_{\frac{\sigma_q\mu_v+\sigma_v\mu_q}{\sigma_v+\sigma_q}, \frac{\sigma_v\sigma_q}{\sigma_v+\sigma_q}}(x)$$

Here, the first term corresponds to the normal distribution $N_{\mu_v,\sigma_v+\sigma_q}(\mu_q)$ and is independent from the integration variable x. Therefore, the first term can safely be written before the integral (as it is a constant). The second term is the pdf of a normal distribution (with some complex values for μ and σ) which always integrates to 1 (when integrating from $-\infty$ to $+\infty$, independently of μ and σ). Therefore, we have

$$\int_{-\infty}^{+\infty} N_{\mu_v,\sigma_v}(x) \cdot N_{\mu_q,\sigma_q}(x)dx$$

$$= N_{\mu_v,\sigma_v+\sigma_q}(\mu_q) \cdot \int_{-\infty}^{+\infty} N_{\ldots}(x)dx = N_{\mu_v,\sigma_v+\sigma_q}(\mu_q)$$

The lemma allows us to calculate the probability that q and v correspond to the same data object by using μ_q as exact feature vector while increasing σ_v by σ_q. Thus, we have reduced the more general case to the easier case that one of the objects is exact and the other is a pfv. To calculate the $p(q|v)$, we have to combine all probabilities for all features and than again apply the rule of Bayes:

$$p(q|v) = \prod_{i=1}^{d} p(q_i|v_i)$$

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

Employing this solution, we can give general algorithms for probability-threshold queries and k-maximum probability queries over a set S of probabilistic feature vectors. For the probability-threshold query, we first have to scan over S to determine the sum of the probability densities of all objects in S, i.e. the total probability. Afterwards, a second scan determines the actual probability $P(v|q)$ for each $v \in S$ and reports those with a probability above the threshold P_θ. For the k-MLIQ, a single scan over the database is sufficient, keeping those k objects (among all objects that have been processed so far) in a local list which have the highest probability density.

For the set S, we can use the whole database DB. In this case, we operate on top of a sequential scan of the database. As an alternative, we can also use a subset of DB which has been generated by a filter step, e.g. an appropriate index structure.

Properties. We conclude this section by briefly summarizing some properties of our solution (without a formal proof) in order to substantiate that the solution agrees with the intuitive requirements of the identification problem.

1. The sum of the probabilities of all retrieved objects of a TIQ or k-MLIQ cannot exceed 100%.
2. To obtain a high identification probability it is required that both database and query objects have a small uncertainty (σ_q, σ_v) and a high compliance of the observed features ($\mu_q \approx \mu_v$), i.e. the Gaussians must have a high overlap and must be steep. Whenever we increase the uncertainty of database or query object (or both), the identification probability will decrease.
3. For very high uncertainty ($\sigma \rightarrow \infty$) of the query or a database object (or both) our model becomes maximally indifferent, i.e. the identification probability corresponds to $1/n$ where n is the number of all possible objects.
4. If the Gaussian of a database object and that of the query object are quite disjoint, the identification probability is close to 0. Only in this case, it is possible that the identification probability slightly increases (up to $1/n$, see above) with increasing uncertainty because when increasing the uncertainty, the degree to which the object can be certainly *excluded from identification* decreases in this case.

5 The Gauss-Tree

In the previous section, we have defined our basic notions of probabilistic feature vectors and queries on top of a set of such pfv. Derived from these basic definitions, we have introduced the basic algorithm for query processing on top of a sequential scan over an unordered file of pfv. The runtime complexity of these algorithms is linear in the number of stored objects. In the context of a large database, this is not acceptable, and we are now going to define the Gauss-tree, a suitable index structure improving k-maximum probability and probability-threshold queries on top of pfv.

5.1 Structure of the Gauss-Tree

The Gauss-tree is a balanced tree from the R-tree family. In contrast to the other index structures from this family,

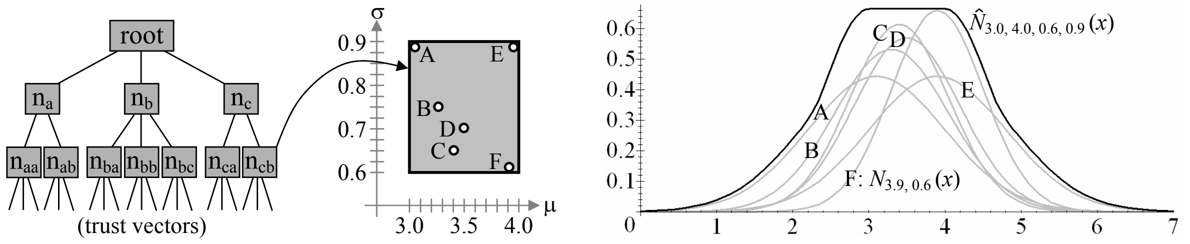

Figure 2. A 3 level Gauss-tree.

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 Gaussian. The structure of the index is inherited from the R-tree family which facilitates the integration into object-relational database management systems.

Definition 4 (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 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 feature value $[\check{\mu_i}, \hat{\mu_i}]$ and every uncertainty value $[\check{\sigma_i}, \hat{\sigma_i}]$ as well as the address of the child node.*
- *All leaf nodes are at the same level.*

In Figure 2, we see an example of a Gauss-tree consisting of 3 levels. In the middle, we have depicted the minimum bounding rectangle of a leaf node for one of the probabilistic features. This minimum bounding rectangle allows to store feature values between $\check{\mu} = 3.0$ and $\hat{\mu} = 4.0$ and uncertainty values between $\check{\sigma} = 0.6$ and $\hat{\sigma} = 0.9$. A few sample pfv which are stored in this data page are also depicted. The Gaussian functions (probability density functions, pdfs) which correspond to these pfv are also shown on the right side of Figure 2 in gray lines.

For query processing, we need a conservative approximation of the probability density functions which are stored on a page or in a certain subtree. Intuitively, the conservative approximation is always the maximum among all (possible) pdf in a subtree. This maximum can be efficiently derived from the minimum bounding rectangle. In Figure 2, the maximum function which has been derived from the depicted minimum bounding rectangle is shown on the right side using a solid black line. As a formula, the approximating pdf $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ is given as:

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

Since we assume independence in the uncertainty attributes, we can safely determine $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ in each dimension separately. Please note that $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ is not really a probability density function as it does not integrate to 1 for the whole data space. It is the conservative approximation of some probability density functions.

5.2 Query Processing

For efficient query processing, 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 2 *The conservative approximation $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ of the probability density functions stored in a data page can be exactly computed by the following piecewise function:*

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

Proof 2 *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 $x \leq \check{\mu}$ and monotonically decreasing for all $x \geq \hat{\mu}$. Therefore, for a given x in the quadrants (I) to (III), the gaussian function which is maximal among all possible functions*

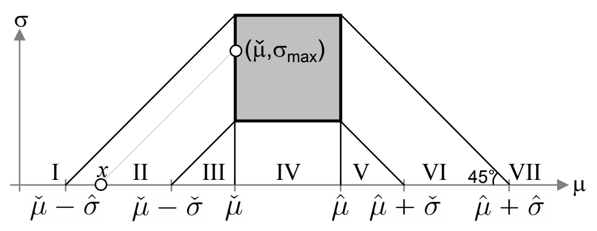
σ
(μ̆,σmax)
I
x
II
III
IV
V
VI
45°
VII
μ
$\check{\mu}-\hat{\sigma}$
$\check{\mu}-\check{\sigma}$
$\check{\mu}$
$\hat{\mu}$
$\hat{\mu}+\check{\sigma}$
$\hat{\mu}+\hat{\sigma}$

Figure 3. The different sectors used to calculate $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$.

$N_{\mu,\sigma}(x), \mu \in [\check{\mu}, \hat{\mu}], \sigma \in [\check{\sigma}, \hat{\sigma}]$ 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.

For query processing we will also need a lower bound $\check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ for the stored Gaussian functions corresponding to the probabilistic feature vectors. This is defined by the following minimum:

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

It can be efficiently computed by considering only 4 Gaussian functions as stated in the following lemma:

Lemma 3 *The lower bound $\check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ for all distance functions stored in a page given by the limits $(\check{\mu}, \hat{\mu}, \check{\sigma}, \hat{\sigma})$ can be computed by:*

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

Proof 3 *When varying μ and σ in our function $N_{\mu,\sigma}(x)$ and fixing x, we observe only one local maximum and no local minimum (and no singularities). Therefore, the global minimum for the restricted function is at one of the four corner points of the rectangle delimited by $(\check{\mu}, \hat{\mu}, \check{\sigma}, \hat{\sigma})$. The four possible minima are tested.*

Note that an even easier method is possible because it is very easy to decide whether the minimum is at $\check{\mu}$ or at $\hat{\mu}$ due to symmetry. All these methods have a constant time complexity.

Later, we will also need the approximation for the sum of all Gaussian functions which are stored in a data node or subtree. For this approximation, we consider the number of objects stored in the subtree n and apply:

$$n \cdot \check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x) \le \sum_{t\in node} N_{\mu_t,\sigma_t}(x) \le n \cdot \hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$$

The accuracy of the approximation of the sum is bounded by:

$$n \cdot (\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x) - \check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x))$$

In our system, a query is defined by a probabilistic feature $q = (\mu_q, \sigma_q)$. The conservative approximations of the maximum, minimum, and sum can be determined analogously to Section 4 by the following equations:

- $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(q) = \hat{N}_{\check{\mu},\hat{\mu},\check{\sigma}+\sigma_q,\hat{\sigma}+\sigma_q}(\mu_q)$
- $\check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(q) = \check{N}_{\check{\mu},\hat{\mu},\check{\sigma}+\sigma_q,\hat{\sigma}+\sigma_q}(\mu_q)$
- etc.

Note that although we have shown in this section only the univariate case, it is very easy to extend all these formulas for the multivariate case because the individual univariate densities can be multiplied as we assume independence among the σ_i. This is also true for the lower and upper bounding pdf $\check{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ and $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ and for the sum approximation. Now we can provide the algorithms for our query types defined in Section 3 on top of the Gauss-tree.

5.2.1 k-Most-Likely Identification Query

A most-likely identification query (MLIQ) reports the object for which the probability-based similarity is maximal. For the Gauss-tree, we give an algorithm operating on top of a priority queue [9]. The algorithm maintains a priority queue of pointers to some of the nodes (called *active nodes*) of the tree. The elements in the queue are ordered by the value of the approximation function evaluated for the query pfv. Let a be a node of the tree with a.appx $= (\check{\mu}_i, \hat{\mu}_i, \check{\sigma}_i, \hat{\sigma}_i, 1 \leq i \leq d)$ the μ and σ bounds associated to node a. Then the priority attribute a.prio of node a in the queue is defined as follows:

$$a.\text{prio}(q) = \hat{N}_{a.appx}(q) = \prod_{1 \leq i \leq d} \hat{N}_{\check{\mu}_i, \hat{\mu}_i, \check{\sigma}_i + \sigma_{q,i}, \hat{\sigma}_i + \sigma_{q,i}}(\mu_{q,i})$$

Intuitively, this ordering key $\hat{N}_{q.appx}$ corresponds to the maximum (relative) probability that one of the Gaussian functions stored in node a could yield when inserting the probabilistic query vector q. The top element of the queue is the node with maximum priority. Initially, the queue contains only the root. The algorithm runs in a loop which removes the top element from the queue, loads the corresponding node from disk (if not in cache), and re-inserts pointers to the children (ordered by their priority attribute) into the queue. The algorithm keeps a *candidate* object in a variable which is the maximum pfv that has been seen so far by the algorithm in any of the leaf nodes. The algorithm stops when a probabilistic feature vector $v = (\mu_1, \sigma_1, \ldots)$ has been found for which the relative probability exceeds that of the top element t of the queue, with t.appx= $(\check{\mu}_i, \hat{\mu}_i, \check{\sigma}_i, \hat{\sigma}_i, 1 \leq i \leq d)$:

$$\prod_{1 \leq i \leq d} N_{\mu_i, \sigma_i + \sigma_{q,i}}(\mu_{q,i}) > \prod_{1 \leq i \leq d} \hat{N}_{\check{\mu}_i, \hat{\mu}_i, \check{\sigma}_i + \sigma_{q,i}, \hat{\sigma}_i + \sigma_{q,i}}(\mu_{q,i})$$

For k-MLI queries we have to maintain the set of k probabilistic feature vectors of maximal probability that have been found so far (the *candidate set*). The algorithm can safely stop now if all pfv in the candidate set have probabilities higher than the top element of the priority queue of the active nodes.

The pseudo code is given in Figure 4.

```
PriorityQueue kMLI_Query(int k, Page root, Point query) {
  // descending priority queues with k entries
  PriorityQueue candidates = new PriorityQueue(descending, k);
  // descending priority queues
  PriorityQueue activePages = new PriorityQueue(descending);
  //init
  activePages.put(root, MAX_REAL);
  //traverse Gauss-tree
  do {
     Page current = activePages.removeFirst();
     if(current is a data page) {
        for each vector in current {
           probability = calcProbability(vector, query);
           candidates.put(vector, probability);
        }
     } else {
       Page successors [] = current.getSuccessors();
       for each s from successors {
          probability = calcMaxProbability(s, query);
          activePages.put(s, probability);
       }
     }
  } while (activePages.isNotEmpty() &&
           candidates.getLastProbability ()
           < activePages.getFirstProbability());
  return candidates;
}
```

Figure 4. Pseudo Code for the k-MLIQ.

5.2.2 Determining the Result Probability

The algorithm in Section 5.2.1 is able to determine those k elements having the highest probability with respect to the query object, but it is not able to determine the actual value of the probability. The reason is, that the stored Gaussian functions are only relative probabilities. These must be contrasted to the sum of the relative probabilities (theoretically) of *all* other Gaussian functions in the database, as discussed in Section 3:

$$P(t|q) = \frac{p(q|t)}{\sum_{s \in DB} p(q|s)}$$

For pages which are far away from the query point, these relative probabilities (and also their approximations) are close to zero. Therefore, not *all* database objects need to be examined in order to determine the true denominator of this formula with sufficient accuracy.

We modify our algorithm of Section 5.2.1 in the following way:

- Whenever a leaf node is accessed, the corresponding pfv are examined and summed up for the total probability.
- Additionally, we maintain the upper and lower bounds for the impact of the objects which are stored in subtrees which have not yet been examined.
- The parent nodes of all subtrees which have not yet been examined are stored in the priority queue. Therefore, the upper and lower bounds are always updated whenever the top element is taken out of the queue and when the child nodes are re-inserted.
- The lower and upper bounds of the part of the sum which is caused by a single node in the tree in which n entries are stored, is given by $n \cdot \check{N}_{\check{\mu}, \hat{\mu}, \check{\sigma}, \hat{\sigma}}(q)$ and $n \cdot \hat{N}_{\check{\mu}, \hat{\mu}, \check{\sigma}, \hat{\sigma}}(q)$, respectively.
- The algorithm stops when both of the following conditions hold:

- The k pfv of highest probability are determined (i.e. all candidates have higher probabilities than the top element of the queue)
- The upper and lower bounds of the sum are close enough together to guarantee that the result is exact for all k answers according to user's specification of exactness (e.g. by a certain number of digits)

5.2.3 Probability Threshold Queries

This algorithm is similar to that of Section 5.2.2 with the difference that an unknown number of possible answer objects is maintained. An object must be stored in that set until it is guaranteed (according to the *lower* bound of the denominator) that the object has a probability which is below the specified threshold. The algorithm can safely stop when for all objects in the answer set it is guaranteed (according to the *upper* bound of the denominator) that they are safely above the specified threshold. We need both a lower and upper approximation of the Gaussian functions stored in a node.

If the user additionally specifies to report the actual probabilities of the answer elements at a specified accuracy, the algorithm may have to access more pages from the priority queue until all probabilities are known with sufficient certainty, like in Section 5.2.2

The pseudo code of our method for the probability threshold query is given in Figure 5.

5.3 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 this case, $\hat{N}_{\dots}(x)$ is narrow, and unnecessary page accesses can be avoided. In contrast, if the node also contains pfv with a high variance then a small range of μ will not help much either because $\hat{N}_{\dots}(x)$ 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 (e.g. k-MPQ) 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. Additionally, our analysis gives a formal justification for the

```
PriorityQueue TI_Query(Page root, Point query, float t) {
  real minSum, maxSum, sum = 0;
  PriorityQueue candidates = new PriorityQueue(desc);
  PriorityQueue activePages = new PriorityQueue(desc);
  //init
  activePages.put(root, MAX_REAL);
  minSum += root.minProb*root.size;
  maxSum += root.maxProb*root.size;
  //search
  do {
    Page current = activePages.removeFirst();
    minSum -= current.minProb*current.size;
    maxSum -= current.maxProb*current.size;
    if(current is a data page) {
      for each vector in current {
        probability = calculateProbability(vector, query);
        candidates.put(vector, probability);
        sum += probability;
      }
    } else {
      Page  successors [] = current.getSuccessors();
      for each s from successors {
        probability = calculateMaximalProbability(s, query);
        activePages.put(s, probability);
        minSum += s.minProb*s.size;
        maxSum += s.maxProb*s.size;
      }
    }
    //delete unnecessary candidates
    while(candidates.getLastProbability()/(minSum+sum) < p)
      candidates.removeLast();
  } while (activePages.isNotEmpty()
      && activePages.getFirstProbability()/(minSum+sum)<p);
  //calculate final probabilities
  for each c in candidates {
    prev = candidates.getProbability(c);
    candidates.updateProbability(c,  prev/(maxSum+sum));
  }
  return candidates;
}
```

Figure 5. Pseudo Code for the TIQ.

strategy. The 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).

The split decision must minimize the probability of a node to be accessed for an arbitrary query. This probability is proportional to the integral of the hull curve:

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

The integral can be determined for each probabilistic feature separately. The computation of the integral is straightforward. Remember the case analysis of lemma 2. 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.

6 Experimental Evaluation

For our experimental evaluation, we implemented the Gauss-tree and all compared methods in Java. All experiments described below were performed on a workstation that is equipped with two AMD Opteron 1.8 GHz processors and 8 GB main memory. We used up to 50 MByte as database cache which was cold started before each experiment. In our experiments, we compare the effectiveness and efficiency of the proposed solution for handling uncertain data on two different data sets.

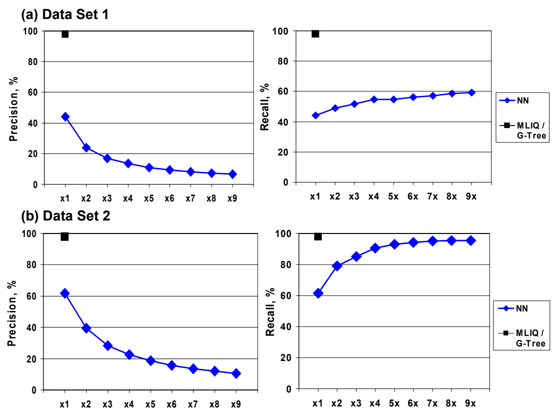

Figure 6. Precision and Recall of 3-NN query on conventional feature vectors and 3MLIQ on pfv.

Data set 1 consists of 10,987 27-dimensional color histograms of an image database. To describe these images as probabilistic feature vectors, we complemented each dimension with a randomly generated standard deviation. A total number of 100 objects was randomly selected and new observed mean value was generated w.r.t. the corresponding Gaussian. For these queries, new standard deviations were randomly generated. To additionally test our method on a larger data set, we randomly generated 100,000 probabilistic feature vectors in a 10-dimensional feature space along with corresponding σ values (data set 2). For data set 2, 500 query vectors were selected and modified as described above.

To demonstrate that ordinary similarity search on feature vectors using Euclidean distance is not sufficient for probabilistic queries on imprecise data, our first experiment compares precision and recall for both methods. To compare the performance of the Gaussian uncertainty model to ordinary similarity search, we processed an MLIQ on the Gauss-tree and a nearest neighbor query on the mean values. For each query, we measured precision and recall. The recall is the percentage of retrieved and correct answers among all objects in the database that are correct and precision is the percentage of the retrieved and correct objects among all objects in the result set. For NN queries and MLIQ, both measures are the percentage of queries that retrieved the

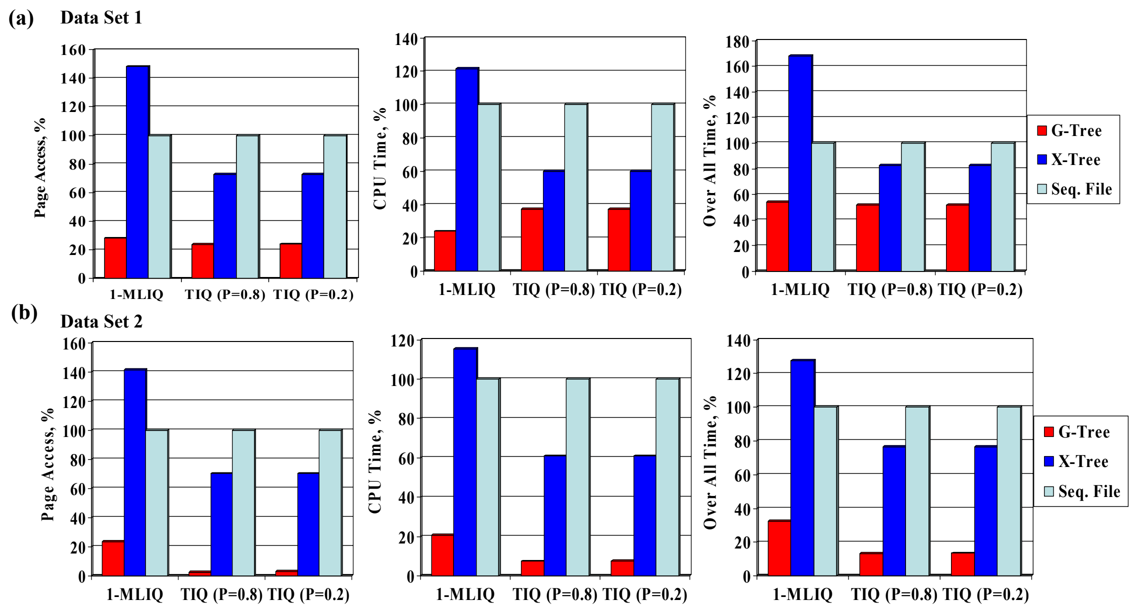

Figure 7. Performance of sequential scan, X-tree on hyper-rectangle approximations of pfv and Gauss-tree on data set 1 and data set 2 (see text for details).

correct object.

Figure 6 compares precision and recall for both methods. The MLIQ achieved a precision and recall of 98% for data set 1 and 99% for data set 2. Thus, our new query, based on the Gaussian uncertainty model, achieved almost optimal results in this experiment. On the other hand, the NN query displayed only a precision of 42% for data set 1 and 61% for data set two. Thus, ordinary similarity search seems not to be suited well for handling uncertain data corresponding to a Gaussian error. To show that ordinary similarity search cannot improve its performance by using larger result sets, we increased the number of retrieved objects for the nearest neighbor query which increases the recall but decreases the precision. For data set 1 the recall did not significantly increase. Even for a result set being 9 times bigger than necessary the recall reached only a value of 60%. For data set 2 the recall could be improved to 97% when using more than 6 times the size of the result set that is necessary. However, due to the dependency between precision and recall, the precision dropped to only 18%. Thus, the right selection of k cannot compensate for the missing handling of uncertainty.

In the next set of experiments, we compare the efficiency of query processing when using the Gauss-tree to the basic solution of a sequential scan over the complete database. To additionally compare to a more sophisticated method, we use an X-tree to store rectangular approximation of each pfv. To derive these approximations, we calculate the 95% quantiles in each dimension, i.e. we determine the interval around the mean value of a Gaussian that contains a random observation with a probability of 95%. By combining these intervals to a hyperrectangle, we generate a good approximation for each pfv. To process a MLIQ with this method we first calculate an approximation for the query pfv. Afterwards, we use the X-tree to determine a candidate set consisting of all approximations that intersect with the query approximation. To find out the final result the candidate set is refined by calculating the exact probabilities. Let us note that this method does not offer exact results with respect to Gaussian uncertainty model because the used approximations allow false dismissals. However, using this method, we observed a precision and recall that was only slightly below the values we observed for the Gauss-tree.

Figure 7 displays the page accesses, cpu time and complete query time for an MLIQ and two TIQ ($P_\theta = 0.2$ and $P_\theta = 0.8$) on both data sets. All values are given in percent to the values of the sequential scan using the Gaussian uncertainty model. For data set 1, the Gauss-tree was able to reduce the page accesses as well as the CPU time up by a factor of 4.2 compared to the sequential scan for all three query types. Though, the all over time suffered from additional seeks on the hard disc, the Gauss-tree was still able to improve query processing by at least 46% for all three types of queries. For data set 2, the Gauss-tree achieved a speed up of 4.3 with respect to the number of accessed pages and of 4.8 for the cpu time of the MLIQ. For the TIQ, it even achieved to improve the page accesses by a factor between 35.7 and 43.2 of the page accesses of the sequential scan and the cpu time by a factor of 13.2 of the cpu time of the sequential scan. The speed up of the all over query time was between 3.1 for the MLIQ and about 7.5 for both TIQ. Thus, the Gauss-tree offered a significant improvement of the efficiency compared to the sequential scan. The X-tree storing rectangular approximations on the other hand did not offer any speed up against the sequential scan for MLIQ. Though it achieved some improvement for the TIQ, it was only capable to decrease the all over time of both queries by 17.3% for data set 1 and 23.2% for data set 2. Thus, it did not offer any real benefit.

7 Conclusions

In this paper, we have introduced the Gaussian uncertainty model for identification queries on inexact, probabilistic feature vectors (pfv). This model extends feature vectors by an additional uncertainty value for each dimension, associating each feature vector to a multivariate Gaussian distribution. To speed up query types such as Threshold Identification Queries (TIQ) or k-Most Likely Identification Queries (k-MLIQ) we propose the Gauss-tree, a balanced index structure from the R-tree family which does not index the Gaussian curves as spatial objects but instead the parameter space of the means and variances of the Gaussians. In our experimental evaluation, we demonstrate the superior quality of the query result when using probabilistic feature vectors as well as the efficiency when using the Gauss-tree. For future work, we plan to investigate the storage of probabilistic feature vectors using paradigms different from hierarchical index structures such as vector approximation. We also plan to parallelize our index structure for a distributed database environment.

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, pp. 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] J. P. Campbell. ”Speaker Recognition: A Tutorial”. Proceedings of the IEEE, Vol. 85, No. 9, 1997.

[4] R. Chellappa, C. Wilson, and S. Sirohey. ”Human and machine recognition of faces: A survey”. Proc. IEEE, 83(5):705–740, 1995.

[5] R. Cheng, D. Kalashnikov, and S. Prakhabar. ”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.

[6] R. Cheng, Y. Xia, S. Prakhabar, 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.

[7] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. ”Fast Subsequence Matching in Time-Series Databases”. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD’94), Minneapolis, MN*, pages 419–429, 1994.

[8] A. Guttman. ”R-trees: A Dynamic Index Structure for Spatial Searching”. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[9] G. Hjaltason and H. Samet. ”Ranking in Spatial Databases”. In *Proc. 4th Int. Symposium on Large Spatial Databases, SSD’95, Portland, USA*, volume 951, pages 83–95, 1995.

[10] K.-I. Lin, H. V. Jagadish, and C. Faloutsos. ”The TV-Tree: An Index Structure for High-Dimensional Data”. *VLDB Journal*, 3(4):517–542, 1994.

[11] R. Mehrotra and J. Gary. ”Feature-Based Retrieval of Similar Shapes”. In *Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria*, pages 108–115, 1993.

[12] F. B. of Investigation. ”The Science of Fingerprints: Classification and Uses”. Washington, D.C., U.S. Government Printing Office., 1984.

[13] T. Seidl and H.-P. Kriegel. ”Efficient User-Adaptable Similarity Search in Large Multimedia Databases”. In *Proc. 23rd Int. Conf. on Very Large Data Bases (VLDB’97)*, pages 506–515, 1997.

[14] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. Face recognition: A literature survey. *ACM Comput. Surv.*, 35(4):399–458, 2003.