Proc. 12. GI-Fachtagung für Datenbanksysteme in Business, Technologie und Web (BTW'07), Aachen, Germany, 2007

Effective and Efficient Indexing for Large Video Databases

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

Abstract: Content based multimedia retrieval is an important topic in database systems. An emerging and challenging topic in this area is the content based search in video data. A video clip can be considered as a sequence of images or frames. Since this representation is too complex to facilitate efficient video retrieval, a video clip is often summarized by a more concise feature representation. In this paper, we transform a video clip into a set of probabilistic feature vectors (pfvs). In our case, a pfv corresponds to a Gaussian in the feature space of frames. We demonstrate that this representation is well suited for accurate video retrieval. The use of pfvs allows us to calculate confidence values for frames or sets of frames for being contained within a given video in the database. These confidence values can be employed to specify two types of queries. The first type of query retrieves the videos stored in the database which contain a given set of frames with a probability that is larger than a given threshold value. Furthermore, we introduce a probabilistic ranking query retrieving the k database videos which contain the given query set with the highest probabilities. To efficiently process these queries, we introduce query algorithms on set-valued objects. Our solution is based on the Gauss-tree, an index structure for efficiently managing Gaussians in arbitrary vector spaces. Our experimental evaluation demonstrates that sets of probabilistic feature vectors yield a compact and descriptive representation of video clips. Additionally, we show that our new query algorithms outperform competitive approaches when answering the given types of queries on a database of over 900 real world video clips.

1 Introduction

Video clips are an important type of multimedia data. Due to recent technical advances, the amount of video data that is available in digital formats as well as the possibility to access and display such video files has increased enormously. Nowadays, it is possible to view complete movies on mobile phones and MP3 players. Another important aspect is that broadcasting videos over the WWW (e.g. in video podcasts) allows to distribute video data to a large number of people while spending minimum effort and budget.

The enormous amount of video clips and movies that is currently available causes a need for database techniques to manage, store and retrieve video data for various applications. In this paper, we focus on the following scenario: Given a database of movies or video clips, we want to retrieve all movies from the database that are likely to match a given set

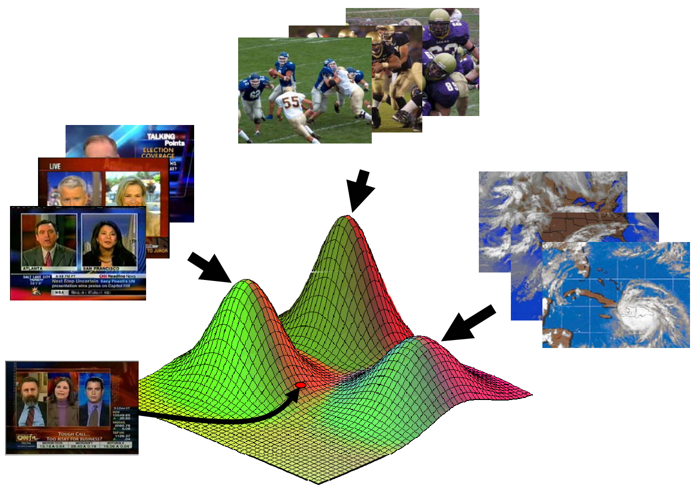


Figure 1: A news video clip summarized as set of probabilistic feature vectors.

of query images. The query images might consist of a continuous image sequence of a scene or might be sampled from the complete movie. For this type of scenario, there are various applications. For example, a company wants to determine if a given video podcast or shared video file contains scenes from any copyright protected movie or video clip. In this scenario, the company would store all of its movies in the database and automatically check if the scenes in the video podcast match any scenes in the database.

Another example is a database of news programs recorded on various days from various tv stations. A user can retrieve all news programs that are likely to contain a given video clip featuring a particular event. Since most news programs use videos which are provided by video news agencies, it is very likely that the news programs dealing with similar topics contain similar news clips. Another application is the detection of commercials in video data recorded from television. In this case, the commercial is the query and the programs are stored in the database. Thus, there are varying applications for this scenario varying from the detection of single scenes to similarity search on complete movies.

From a technical point of view video data consists of a sequence of images (so-called frames) that might be accompanied with some soundtrack. In our approach, we focus on the image part only. To allow similarity search on video clips, each frame is usually represented by a feature vector corresponding to some content based image representation such as color histograms or texture vectors. So-called summarization [ZRHM98, GGM02, CSL99] techniques are used to reduce the enormous number of frames. For summarization, a video is decomposed into shots, i.e. a sequence of frames within a movie showing the same scenario recorded from the same camera position. The images within a shot are usually very similar and thus, the images are usually associated to very similar feature vectors. Therefore, each shot can be summarized by some representative object and only the representative objects are stored in the database. To represent a shot, it is often sufficient to simply take the centroid or mean vector of all feature vectors within the shot. Newer approaches like [IBW$^+$04] represent shots as Gaussian probability density functions (pdf) where each component μ_i of the mean vector is complimented by a variance σ_i^2. We call such feature vectors where each vector component is associated to a variance value *probabilistic feature vector* (pfv). This type of summarization is usually more accurate because the method additionally considers the variance among the summarized feature values. In our new approach, we condense the given video data even more, by representing all similar frames by one Gaussian regardless of the shot they belong to. To conclude, each movie in the database is represented by a *set of probabilistic feature vectors* (pfvs) where each Gaussian represents a set of similar frames.

Our work is focused on similarity search and scene detection in movie databases. To pose a query, a user has to provide a video clip that might comprise a scene in the movie or even the complete movie. The query clip can be transformed into a set of frames, corresponding to a set of traditional feature vectors or a set of probabilistic feature vectors. To use probabilistic (rather than traditional) feature vectors for the *queries* yields advantages as well as disadvantages: extracting a set of frames and determining traditional feature vectors without further summarization might be computationally simpler and less expensive. In contrast, probabilistic feature vectors might represent the information contained in the query in a more concise way. Therefore, we will examine both possibilities.

Furthermore, we develop a method for comparing both types of query representations to objects stored in the database which is based on the likelihood that the query matches the database object. Based on this method, we describe two types of probabilistic queries. The first type is the *set-valued probabilistic threshold query* retrieving all movies matching the given query frames with a likelihood which is higher than a specified threshold value. The second query type is the *set-valued probabilistic ranking query* retrieving the top k movies from the database which are most likely query hits.

Although summarization considerably decreases the size of the representation of each database object, query processing still requires to examine every movie description in the database. Therefore, we will introduce algorithms for query processing that are facilitated by the Gauss-tree [BPS06b], an index structure for probabilistic feature vectors. Let us note that our previous work on the Gauss-tree was focused on querying single objects. In this paper, we introduce techniques for querying set-valued objects which is a more complex problem.

Our main contributions in this paper are:

- A compact representation of a video as sets of probabilistic feature vectors and a method for similarity and partial similarity search based on statistics.
- The specification of two new types of probabilistic queries on sets of probabilistic feature vectors.
- Efficient algorithms for processing these new types of queries on sets of probabilistic feature vectors which are based on the Gauss-tree.

The rest of the paper is organized as follows. Section 2 surveys related topics like content based video retrieval and similarity search using point sets and probabilistic feature vectors. Additionally, the Gauss-tree is introduced as the index structure the query algorithms are based on. In section 3, we will formalize our model and the new types of queries. Section 4 describes the new algorithms for query processing. To demonstrate the quality of our approach to video retrieval and show the superior efficiency of our query algorithms, we provide several experiments on a database of over 900 video clips in section 5. The paper is concluded by section 6 containing a short summary.

2 Related Work

2.1 Video Summarization Techniques.

Since video data consists of large sequences of images or frames, a straightforward feature representation of a movie might contain thousands or even millions of feature vectors. In order to handle such data efficiently, summarization techniques are usually applied to the original data, i.e. the original feature vectors are grouped together and each group is represented by a summarization vector or summarization representative. Then similarity is defined based on these summarizations. Summarizations are usually generated by applying optimization algorithms on feature vectors. They describe a video as a mix of statistical distributions or cluster representatives. The authors of [CSL99] propose an approach for obtaining a compact representation of videos that computes the optimal representatives by minimizing the Hausdorff distance between the original video and its representation. There also exist approaches which apply k-medoid or k-means clustering for the summarization of video clip content [ZRHM98]. In [GGM02], a summarization technique is presented which describes spatial-temporal areas in a sequence of a few dozen frames by mixtures of Gaussian distributions. The authors of [IBW$^+$04] demonstrated that Gaussian mixture models computed from video shots yield higher retrieval precision compared to keyframe-based models. However, to the best of our knowledge, none of these techniques uses an index structure for the pfvs to accelerate query processing.

2.2 Similarity Search Based on Set-Valued Objects

Set-valued objects are usually compared by complex distance measures like [EM97, RB01] allowing similarity queries. However, selecting a suitable distance measure for a particular application is often quite difficult because there exist many different notions of similarity between two sets of feature vectors. Another problem is the understandability of the derived distances. For complex distance measures and large set-valued objects containing hundreds of instances, it is very difficult to understand why the set-valued objects are similar. Finally, employing the proposed distance measures often yields efficiency problems. Since most of the distance measures for set-valued objects are non-metric, employing index structures is not always possible. Additionally, useful filter steps avoiding time consuming distance calculations like in [BKK$^+$03] were introduced for a minority of multi-instance distance measures only. To the best of our knowledge there is so far no query algorithm handling sets of probabilistic feature vectors, instead of ordinary set-valued objects.

2.3 Similarity Search Based on Probabilistic Feature Vectors

In [CKP03] a new uncertainty model is introduced and several new types of queries are described that allow the handling of inexact data. [CXP$^+$04] describes two methods for efficiently answering probabilistic threshold queries that are based on the R-Tree [Gut84]. A probabilistic threshold query returns all data objects that are placed in a given query interval with a probability exceeding a specified threshold value. [TCX$^+$05] introduced the U-Tree for indexing uncertain 2D objects. All these approaches do not handle sets of probabilistic feature vectors and do not apply a Bayesian setting. Thus, the mentioned approaches are rather dealing with data objects having an uncertain location. Besides the mentioned methods for indexing spatially uncertain objects, [DYM$^+$05] introduces existential uncertainty. The idea of this approach is that the existence of each data object is uncertain.

2.4 The Gauss-tree

In [BPS06b], the Gauss-tree is introduced which is an index structure for managing large amounts of Gaussian distribution functions. Additionally, [BPS06b] proposed probabilistic identification queries which are based on a Bayesian setting, i.e. the paper deals with the retrieval of the database objects that explain a given query observation with the highest probability. This setting is more similar to the queries described in this paper. However, the queries in [BPS06b] are based on the assumption that there is exactly one individual object explaining the query object. In our setting a very important aspect is that one query video clip might be contained in several database movies. Another major difference to the approach described in this paper is that [BPS06b] strictly deals with single-valued probabilistic feature vectors. In [BPS06a] the Gauss-tree was extended to handle objects having an uncertain location as proposed in [CXP$^+$04].

Since the Gauss-tree is the index structure our new method is based on, we will now survey the main characteristics of this approach and the processing of single-valued queries. For the Gauss-tree, a single pfv is defined as follows:

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:

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

Let the dimensionality of the data space be d, i.e. our pdf are d-variate Gaussian functions each of which is defined by d pairs of means and standard deviation $(\mu_i, \sigma_i, 1 \leq i \leq d)$. According to this definition our method is based on independent features which is often not given in a given application. However, as in naive Bayes classification, neglecting the dependencies between the dimensions does not necessarily cause a bad retrieval performance. Furthermore, in image data the correlations between the features are more or less an inherent characteristic of the transformation method and not to a given database. Thus, it is possible to use feature transformation techniques like principal component analysis (PCA) to find orthogonal dimensions. The idea of the Gauss-tree is to regard the parameters of each Gaussian as vectors (points) of a $(2 \cdot d)$-dimensional space. The structure of the index is then inherited from the R-tree [Gut84] family, as formalized in the following definition:

Definition 2 (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 $(\mu_i, \sigma_i), 1 \leq i < d$.*
- *An entry of a non-leaf node is a minimum bounding rectangle of dimensionality $2 \cdot d$ 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, pdf) 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) pdfs in a subtree. This maximum can be efficiently derived from the minimum bounding rectangle. In Figure 2, the

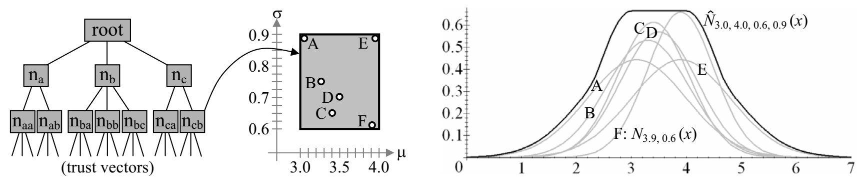


Figure 2: A 3 level Gauss-tree.

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:

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

With a case distinction involving seven different cases, $\hat{N}_{\ldots}(x)$ can be efficiently and analytically computed:

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

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 a set of probability density functions.

Similarly to the other index structures from the R-tree family, the Gauss-tree is constructed by iteratively inserting new objects. A node split operation is performed whenever a node exceeds its defined capacity (M). For the selection of a branch of the Gauss-tree upon insertion of a new object and for the determination of a split dimension, strategies have been proposed which minimize the integral of $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$.

Several query types for databases of Gaussian pfv have been defined. Query objects may either be conventional d-dimensional feature vectors (exact queries) or probabilistic feature vectors (probabilistic queries). Probabilistic queries can be processed like exact queries if the variances of the query are added to the corresponding variances of the pfv stored in the database. The first defined query type is the k-most likely identification query (k-MLIQ) which reports the k objects having maximum probability-based similarity. Given the query vector q, the algorithm accesses the nodes of the Gauss-tree in increasing order of $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$. A priority queue [HS95] is used to support this access order. Query processing stops when k pfv have been retrieved having a higher probability at the query point than the hull function $\hat{N}_{\check{\mu},\hat{\mu},\check{\sigma},\hat{\sigma}}(x)$ of the top page in the priority queue.

In a similar way, probability threshold queries are processed. For this query type, the user specifies the threshold P_Θ of the probability of the query answers rather than the number of answers.

3 Video Retrieval using Probabilistic Feature Vectors

In this section, we will formalize video summarization using sets of probabilistic feature vectors (pfvs) following a Gaussian density function. Additionally, we will provide the probabilistic framework for comparing queries to movies and specify the new types of queries.

As mentioned before, the video part of a movie is a sequence of images which can be transformed into d-dimensional feature vectors $f \in \mathbb{R}^d$. Applying summarization techniques, a video is represented by a set of pfvs. Let us note that there are other notions of pfvs which are based on different density functions, but in this paper the distribution function of a pfv is considered to be Gaussian. Thus, our pfvs are defined as proposed in definition 1.

To represent a movie, we employ a set of pfvs. Each pfv is considered to represent a set of similar frames in the movie. Let us note that a pfv does not necessarily correspond to a single shot. Instead, we summarize similar frames without considering shots first. Additional to each pfv, we consider a weight w_i expressing the average amount of frames represented by the given pfv v_i in the complete movie. Thus, pfvs representing more frames have larger weights than pfvs representing a smaller fraction of the frames. We can now define a movie descriptor as follows:

Definition 3 *A movie descriptor M is a set of pfvs $\{v_1, \ldots, v_k\}$ and a weighting $\{w_1, \ldots, w_k\}$. w_i corresponds to the a priori likelihood that a frame in the movie is described by the pfv v_i. Furthermore, the following condition holds:* $\sum\limits_{i=1}^{k} w_i = 1$

A query is posed by specifying a video clip or only a part of it. To calculate the likelihood that the query is contained in some database object, we first of all have to apply some feature transformation to the query as well. Thus, a query Q can be considered as a set of feature vectors $\{q_1, \ldots, q_l\}$ with $q_i \in \mathbb{R}^d$. To calculate the probability that Q is contained in a movie described by M, we first of all have to derive a probability for a single query frame q_i for being contained in a given pfv $v_j \in M$ having the weight w_j. A pfv corresponds to a density function over $\mathbb{R}^d$. Thus, we can calculate the density of q_i w.r.t. v_i. However, to calculate a probability for a single vector in a continuous space, we would have to integrate over some interval. Since for a single vector this interval converges to 0, the probability of the vector converges to 0 as well. However, since we already observed q_i, we actually do not need to calculate the probability that exactly q_i

occurs in the given video. Instead, we can apply the theorem of Bayes and calculate the conditional probability that q_i belong to v_j under the condition it appeared at all. To formalize this condition, we have to distinguish three cases. First, q_i belongs indeed to v_j. Second, q_i belongs to some other pfv v_k in the same movie M. Finally, q_i is not contained in M but is part of some other movie. To approximate the last case, we specify $H_0(q_i)$ which is modeled by a uniform distribution or the average density of any known pfv for the vector q_i. Additionally, we multiply this density with the number of pfvs in the compared movie descriptor to have a weighting which is equal to the movie descriptor.

Thus, the probability that q_i appears at all is the sum of the probabilities $p(q_i|v_i)$ that q_i belongs to some v_i describing the current movie M and the probability that q_i is not contained in M. The later probability is expressed by $H_0(q_i)$. Formally, we can calculate the probability $P(v_j|q_i)$:

$$P(v_j|q_i) = \frac{w_j \cdot p(q_i|v_j)}{\sum_{\hat{v}\in V} \hat{w} \cdot p(q_i|\hat{v}) + H_0(q_i)}$$

Since a movie is given by a set of pfvs, the probability that a frame q_i is contained in the complete movie described by M, can be computed by summing up the probabilities for each pfv:

$$P(M|q_i) = \sum_{v_j \in M} P(v_j|q_i)$$

Finally, we have to consider all frames $q_i \in Q$ of a query. Thus, we calculate the average probability for any frame in the query q_i for being contained in the given movie descriptor M by:

$$P(M|Q) = \frac{\sum_{q\in Q} P(M|q)}{|Q|}$$

If a query comprises large numbers of frames this method yields performance problems. Thus, we have to reduce the number of frames for the query object as well. If the query must be answered in interactive time, sophisticated summarization techniques cannot be applied. Thus, we propose a simple reduction by considering every ith frame only. If time is less important, summarization by sets of pfvs is applicable. In this case, the query is represented by a movie descriptor itself. For calculating the probability that a movie descriptor M_q describes frames which are contained in the movie described by M, we will proceed as follows. We first of all determine the probability that a query pfv v_q describes the same set of feature vectors as a pfv v_m contained in the movie. This probability can be defined as follows:

The probability density of two Gaussians for describing the same vector can be specified as follows:

$$p(v_q, v_m) = \int_{-\infty}^{+\infty} p(v_q|x)p(v_m|x)dx$$

Having this probability, we can calculate the conditional probability for v_m under the condition of v_q in the following way:

$$P(v_m|v_q) = \frac{w_m \cdot w_q \cdot p(v_q, v_m)}{\sum_{\hat{v} \in M} \hat{w} \cdot w_q \cdot p(v_q, \hat{v}) + H_0}$$

Using this probability, we can proceed as above. The probability for $P(M|M_q)$ is the average probability of $P(M|v_q)$ which is the sum over all $P(v_j|v_q)$ in M:

$$P(M|M_q) = \frac{\sum_{v_q \in M_q} \sum_{v_j \in M} P(v_j|v_q)}{|Q|}$$

Based on these probabilities, we can specify probabilistic queries retrieving any movie in the database having a large enough probability for containing a query video clip. To decide which probability is large enough for being contained in the result set, there are two general approaches. The first is to define a fixed probability threshold, e.g. 80%. Thus, we retrieve all movies containing the specified query frames with a probability of more than 80%. Formally, we can define a set-valued probabilistic threshold query on movie descriptors as follows:

Definition 4 (Set-Valued Probabilistic Threshold Query) ***(SVPTQ)*** *Let DB be a database of movie descriptors, let Q be a set of query frames and let $P_\theta \in [0 \ldots 1]$ be a probability threshold. The answer of a threshold identification query is defined as follows:*

$$SVPTQ_{DB}(Q, P_\theta) = \{M \in DB | P(M|Q) \geq P_\theta\}$$

The second method for deciding containment in the query result is to retrieve the k most likely results. Thus, the threshold is relative to the database content. An example for this type of query is: Retrieve the 5 movies from the database having the highest probability for containing the query scene. We will call this type of query set-valued probabilistic ranking query (SVPRQ). In the following we will formalize SVRCQs:

Definition 5 (Set-Valued Probabilistic Ranking Query)
(SVPRQ) *Let DB be a database of movie descriptors M, let Q be a set of query frames and let $k \in \mathbb{N}$ be a natural number. Then, the answer to a set-valued probabilistic ranking query (SVPRQ) on DB is defined as the smallest set $RQ_k(Q) \subseteq DB$ with at least k elements fulfilling the following condition:*

$$\forall M_a \in RQ_k(Q), \forall M_{db} \in DB \setminus RQ_k(Q) : P(M_a|Q) > P(M_{db}|Q)$$

4 Indexing Summarized Videos

After describing the queries, we are now introducing our solution for efficient query processing based on sets of probabilistic feature vectors.

4.1 Answering Set-Valued Queries

In contrast to searching in a database where each object is represented by a single pfv, our application requires the use of set-valued objects for both the query and the database objects. For query processing, we have to match all the elements of the query representation (being traditional or probabilistic feature vectors) against all the movie descriptors in the database. The difficulty of this task lies in the problem that even if a movie descriptor offers a high likelihood for containing one of the elements of our query, the corresponding movie needs not necessarily to be a likely candidate for containing the complete query. Thus, in order to prune a movie descriptor from the search space, it is necessary to approximate the probability of the complete movie descriptor for matching the complete query.

Our new method for indexing movie descriptors uses a single Gauss-tree for managing all pfvs belonging to any movie descriptor in the database. Each pfv is identified by its movie ID and an additional sequence number identifying the pfv within the movie. To utilize this data structure for answering matching queries, we will describe conservative approximations of the likelihood that the elements of a query Q are described by some movie descriptor being stored in a set of nodes belonging to the Gauss-tree.

Therefore, we will first of all calculate the probability of a query element $q_i \in Q$ that q_i is contained in some movie M descriptor which is completely stored in a set of nodes P:

Lemma 1 *Let Q be a set-valued query, let $P = \{p_1, \ldots, p_m\}$ be a set of nodes in the Gauss-tree T containing the pfvs of a movie Descriptor $M \in DB$. We define the function $maxDense_P(q)$ as follows:*

$$maxDense_P(q) = \max_{p_i \in P} N_{p_i}(q)$$

Then the following condition holds for all $q \in Q$:

$$\forall M \in P : P(M|q) \leq \frac{maxDense_P(q)}{maxDense_P(q) + H_0}$$

Proof 1

$$P(M|q) = \frac{\sum\limits_{v_i \in M} w_i \cdot p(q|v)}{\sum\limits_{v_i \in M} w_i \cdot p(q|v) + H_0(q)} \leq \frac{\max\limits_{p_j \in P} N_{p_j}(q)}{\max\limits_{p_j \in P} N_{p_j}(q) + H_0(q)}$$

$$\Leftrightarrow \sum_{v_i \in M} w_i \cdot p(q|v) \leq \max_{p_j \in P} N_{p_j}(q)$$

$$\Leftrightarrow \sum_{v_i \in M} w_i \cdot p(q|v) \leq \sum_{v_i \in M} w_i \cdot \max_{p_j \in P} N_{p_j}(q)$$

$$= \max_{p_j \in P} N_{p_j}(q) \cdot \sum_{v_i \in M} w_i = \max_{p_j \in P} N_{p_j}(q) \cdot 1$$

Based on this lemma, we can determine the maximum probability for each element q of the query Q of being contained in a movie M which is completely stored in the set of pages P. To employ this lemma for approximating the likelihood of the complete query Q, we must take the average of the conservative approximations over all elements of the query Q. The average of a set of conservative approximations must be a conservative approximation of the average of the exact values. Since each part of the sum in the average of approximations is greater or equal to the exact value, the sum of approximations is greater or equal than the sum of exact values as well. The average is the mentioned sum divided by the number of elements. Therefore, the following condition holds:

$$\forall M \in P : P(M|Q) \leq \frac{1}{|Q|} \cdot \sum_{q_i \in Q} \frac{maxDense_P(q)}{maxDense_P(q) + H_0(q)}$$

Though we can now approximate the probability that Q matches some movie $M \in P$, the approximation is potentially depending on several nodes $p \in P$ at the same time. For ranking and pruning nodes in the query algorithms, we therefore prove the following lemma:

Lemma 2 *Let Q be a set-valued query, let $P = \{p_1, \ldots, p_m\}$ be a set of nodes in the Gauss-tree T containing the pfvs of any movie descriptor $M \in DB$. Then the following condition holds:*

$$\begin{aligned}\forall M \in P : P(M|Q) &\leq \max_{p \in P, q \in Q} \frac{N_p(q)}{N_p(q) + H_0(q)} \\ &= \max_{p \in P} maxProb(Q, n)\end{aligned}$$

Proof 2

$$\begin{aligned}\forall M \in P : P(M|Q) &\leq \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{\max\limits_{p \in P} N_p(q)}{\max\limits_{p \in P} N_p(q) + H_0(q)} \\ &\leq \frac{|Q|}{|Q|} \cdot \max_{q_i \in Q} \frac{\max\limits_{p \in P} N_p(q)}{\max\limits_{p \in P} N_p(q) + H_0(q)} \\ = \max_{q \in Q} \max_{p \in P} \frac{N_p(q)}{N_p(q) + H_0(q)} &= \max_{p \in P, q \in Q} \frac{N_p(q)}{N_p(q) + H_0(q)}\end{aligned}$$

We can now approximate the probability $P(M|Q)$ that M is completely stored in the set of nodes P on the basis of a single node p_{max} where p_{max} is the node p maximizing $maxProb(Q, p)$. An important property of this approximation is that it can be used to rank the access order of the nodes in the Gauss-tree for query processing. Additionally, we will employ this lemma for pruning unnecessary pages and terminate our queries.

Our algorithms employ two data structures. The first is a priority queue containing the nodes of the Gauss-tree that have not been examined yet. The priority is ranked with respect to $maxProb(Q, p)$ in descending order. Due to Lemma 2, $maxProb(Q, p)$ yields an upper bound of the probability of a movie descriptor to be completely stored in the remaining nodes of the tree. Additionally, $maxProb(Q, p)$ can be considered as the maximum probability for all query elements that are yet unknown.

The above lemmas describe the case that there is a set of the nodes which are guaranteed to contain the complete set of considered movie descriptors. However, during query processing we will encounter the case that we already retrieved some pfvs for a movie M, but there are still some $v \in M$ which are stored in the part of the Gauss-tree that has not been examined yet. For those movie descriptors, we have to store the already known densities in the so-called candidate table until the complete set of pfvs is retrieved. Each entry in the candidate table corresponds to a movie descriptor. For each movie stored in the candidate table, we additionally store the sum of the densities for each query element q and each density function v_i that has been retrieved so far. Let us note that each density $p(q|v_i)$ in each sum is weighted with w_i which is the weight of the pfv v_i in the descriptor M. Finally, we store the number of all already retrieved density functions for each movie descriptor M. Based on this data and the current $maxProb(Q, p)$ on the top of our priority queue, we can also approximate the density of any partly known movie descriptor. The approximation is formulated in the following lemma:

Lemma 3 *Let M be a partially retrieved movie descriptor, $A \subset M$ be the set of already known pfvs with weight w_a and let $B \subset M$ be the still unknown elements of M. Furthermore, let P be the set of node in the Gauss-tree P containing B. We define the function $partDensity_A(q)$ as follows:*

$$partDensity_A(q) = \sum_{v_i \in A} w_i \cdot p(q|v_i) + (1 - \sum_{v_i \in A} w_i) \cdot maxDense_P(q)$$

Then, the following condition holds:

$$P(M|q) \leq \frac{partDensity_A(q)}{partDensity_A(q) + H_0(q)}$$

Furthermore, we can state for the complete query Q:

$$P(M|Q) \leq \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{partDensity_A(q)}{partDensity_A(q) + H_0(q)}$$

Proof 3 *The proof is analogue to the proof of lemma 2.*

4.2 Set-Valued Probabilistic Threshold Query

In our first query, we have a fixed global probability threshold P_Θ which can be employed to decide whether a movie is part of the result set. We will now explain our algorithm for

```
SVPTQ(Query Q, float $P_\Theta$)
  activePages := new PriorityQueue(descending)
  candidateTable := new CandidateTable()
  result := new List()
  pruned := new List()
  $activePagesQueue$.insert(root, 1.0)
  DO
    $aktNode$ = activePages.removeFirst()
    IF $aktNode$ is a directory node THEN
      FOR each $node$ in $aktNode$ DO
        activePages.insert($node$,$maxProb(Q, node)$)
      END FOR
    END IF
    IF $aktNode$ is a data node THEN
      FOR each $pfv$ in $aktNode$ DO
        IF pfv.MovieID in pruned THEN
          CONTINUE
        END IF
        candidateTable.update(pfv.MovieID, pfv(Q))
        candidateEntry := candidateTable.get(pfv.MovieID)
        IF candidateEntry.isComplete THEN
          IF candidateEntry.probability(Q) $\geq P_\Theta$ THEN
            result.add(pfv.MovieID)
          END IF
          candidateTable.delete(pfv.MovieID)
        ELSE
          IF andidateEntry.approximation(Q) $\leq P_\Theta$ THEN
            pruned.add(pfv.MovieID)
            candidateTable.delete(pfv.MovieID)
          END IF
        END IF
      END FOR
    END IF
  WHILE((not candidateTable.isEmpty
     or activePages.topProbability $> P_\Theta$)
    and not activePages.isEmpty())
  RETURN $result$;
```

Figure 3: Pseudocode of Set-Valued Probabilistic Threshold Query.

processing SVPTQs using the Gauss-tree. The pseudo code of this algorithm is displayed in Figure 3. The algorithm starts by reading the root node of the Gauss-tree. For each node p being a child node of the root, we now calculate $maxProb(Q, p)$ and insert the nodes into the priority queue which is sorted in descending order. Afterwards, the algorithm enters its main loop which iterates until the priority queue is empty. Additionally, the algorithm terminates if we can guarantee that there cannot be any movie descriptor left matching the given query Q with a likelihood larger than P_Θ. In each step, the algorithm removes the top element of the queue. If the element is a node, it is loaded and pointers to its child nodes are inserted into the priority queue, ranked by $maxProb(Q, p)$. If the top element of the queue is a pfv, we check if there is already an entry in the candidate table corresponding to the movie descriptor M of the pfv. If not, we insert a new entry into the candidate table. In both cases, we can update the sum for each query element for the movie descriptors in the candidate table. If the current entry for the movie descriptor M is complete, i.e. all of its pfvs have been retrieved, we can calculate the likelihood. If this likelihood is larger than t, we can add M to the result set. Finally, the entry for M is removed from the candidate table.

```
SVPRQ(Query Q, integer $k$)
  activePages := new PriorityQueue(descending)
  resultQueue := new PriorityQueue(ascending)
  candidateTable := new CandidateTable()
  pruned := new List()
  $activePagesQueue$.insert(root, 1.0)
  DO
    $aktNode$ = activePages.removeFirst()
    IF $aktNode$ is a directory node THEN
      FOR each $node$ in $aktNode$ DO
        activePages.insert($node$,$maxProb(Q, node)$)
      END FOR
    END IF
    IF $aktNode$ is a data node THEN
      FOR each $pfv$ in $aktNode$ DO
        IF pfv.MovieID in pruned THEN
          CONTINUE
        END IF
        candidateTable.update(pfv.MovieID, pfv(Q))
        candidateEntry := candidateTable.get(pfv.MovieID)
        IF candidateEntry.isComplete THEN
        prob := candidateEntry.probability(Q)
          IF prob$\geq$ resultQueue.topProbability THEN
            IF resultQueue.size = $k$ THEN
              resultQueue.removeFirst
            END IF
            resultQueue.add(pfv.MovieID,prob)
          END IF
          candidateTable.delete(pfv.MovieID)
        ELSE
          IF candidateEntry.approximation(Q) $\leq$
            resultqueue.topProbability THEN
            pruned.add(pfv.MovieID)
            candidateTable.delete(pfv.MovieID)
          END IF
        END IF
      END FOR
    END IF
  WHILE((not candidateTable.isEmpty
    or activePages.topProbability $>$ resultqueue.topProbability)
    and not activePages.isEmpty())
  RETURN $result$;
```

Figure 4: Pseudocode of Set-Valued Probabilistic Ranking Query

If the movie descriptor M is not complete after updating the priority queue, we approximate the current maximum likelihood of M and Q. If the conservative approximation is smaller than t, we can exclude M from the result set. Thus, we store the ID of M in a separated pruning list and delete its entry from the candidate table. If we later encounter a pfv belonging to M, we can safely skip its computation after checking the pruning list. Our algorithm terminates if $maxProb(Q, p)$ for the top element of the priority is smaller than P_Θ. Additionally, we have to continue processing until the candidate table is empty, to make sure that the result is complete.

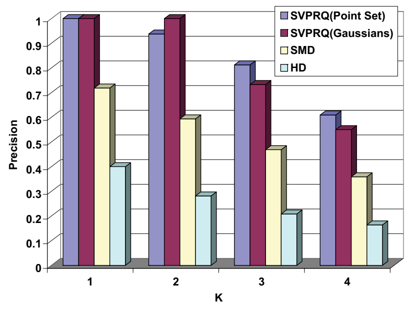
4.3 Set-Valued Probabilistic Ranking Query

The second query type proposed in this paper are SVPRQs. For SVPRs the minimum probability for a result depends on the movie having the k highest probabilities for containing the query set. The idea of the algorithm is quite similar to the previous algorithm. However, for this type of query, we need a second priority queue storing those k movies which currently have the largest probabilities for containing Q. We will sort this second priority queue in ascending order and refer to it as result queue. The pseudo code for this algorithm is displayed in Figure 4. We start again by ordering the descendant nodes of the root page w.r.t. $maxProb(Q, p)$. Afterwards we enter the main loop of the algorithm and remove the top element of the queue. If this element is a node, we load its child nodes. If these child nodes are nodes themselves, we determine $maxProb(Q, p)$ and update the priority queue. If the child nodes are pfvs, we check the candidate table for corresponding movie descriptor M and insert a new descriptor, in the case that there is not already a descriptor for the movie M. Afterwards, we can update the candidate table as mentioned before. If a movie descriptor M has been read completely, we can delete it from the candidate table and compare its probability $P(M|Q)$ to the probability of the top element of the result queue, i.e. the movie descriptor encountered so far having the k highest probability. If the probability of M is higher than that of the top element, we need to add M to the queue. However, to make sure that we do not retrieve more than k elements, we have to check the size of the result queue. If there are already k elements, we have to remove the top element before inserting M. In the case, that the entry in the candidate table does not contain the complete information about M yet, we still can calculate a probability estimation and compare it to the top element of the result queue. If $P(M|Q)$ is smaller than the k highest probability in the result queue, we can guarantee that M is not a potential result. Thus, M is deleted from the candidate table and stored in our list for excluded movie descriptors. The algorithm terminates if the top of the priority containing the remaining notes provides a lower value than the top of the result queue and the candidate table is empty.

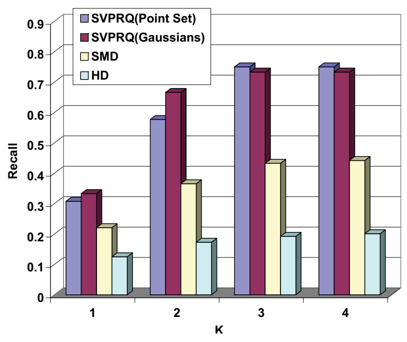
5 Experimental Evaluation

5.1 Testbed

All experiments were performed on a workstation featuring a 2.2 GHz Opteron CPU and 8GB RAM. All algorithms are implemented in Java 1.5. We evaluated our SVTCQ, SVRCQ and their comparison partner using a database of 902 music video clips recorded from various TV stations. The average length of a video clip within our collection is 4 minutes and 6 seconds. We extracted the image representations of the videos on a per-frame basis, i.e. we generated 25 features/second for PAL and 30 features/second for NTSC videos. From each image, we extracted a color histogram. For the color histogram, we used the HSV color space which was divided into 32 subspaces, 8 ranges of hue and 4 ranges of saturation.



(a) Precision.



(b) Recall.

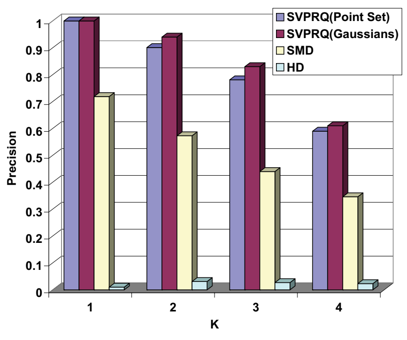
Figure 5: Precision and recall achieved on similarity search by SVPRQ and its comparison partners on complete video retrieval.

In order to obtain the summarization for each video clip, we applied the EM clustering algorithm. The EM clustering provided us with approximately 100 multivariate Gaussians per video clip. In our experiments, we performed video similarity search. As setup step, we picked 40 query videos from our database and manually selected a set of videos which are similar to the query videos.

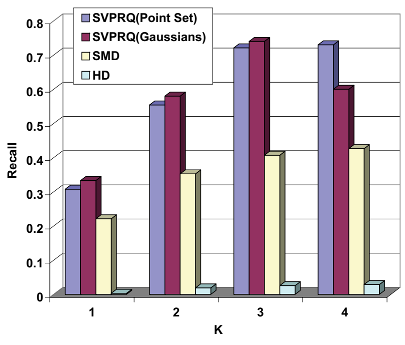
To generate queries, we employed two methods for collecting query frames. The first method tried to capture the complete video clip. Thus, we sampled every 50th frame from the complete clip to derive a representative sample of frames. The second method simulated queries which are posed by giving only a scene or shot from the video. Therefore, we sampled a random interval from the sequence of all frames in the video corresponding to about 500 frames, i.e. 20 seconds. For this type of query, we used every 10th frame of the query interval, i.e. we used 50 frames per query. Additional to these queries, we also generated queries which are represented by sets of probabilistic feature vectors. For representing the complete video, we again employed EM clustering for 100 clusters on the complete set of frames in one video clip. For the queries on the scenes, we clustered the 500 frames, deriving 5 Gaussians.

To have comparison partners for retrieving videos on sets of ordinary feature vectors, we generated a database containing color histograms for all frames of every video clip in our test set. We employed two well-established distance measures for set-valued objects to pose queries to this database, the Hausdorff(HD) distance and the sum of minimum distances (SMD)[EM97]. For these methods we could only use the query consisting of sets of feature vectors.

Our first set of experiments examined the precision and recall of video retrieval for all 4 types of generated queries. Therefore, we performed kNN queries for our comparison partners and SVPRQ for the methods proposed in this paper. The result for the queries on the complete video clips is displayed in Figure 5. As a first result it can be seen that



(a) Precision.



(b) Recall.

Figure 6: Precision and recall achieved on similarity search by SVPRQ and its comparison partners using scene retrieval.

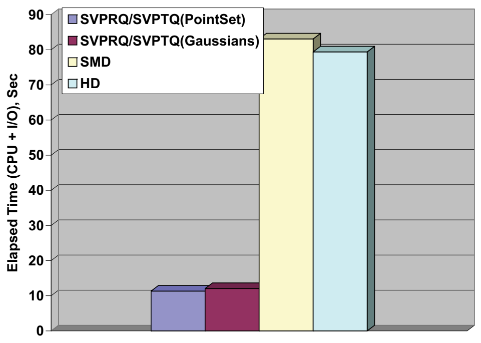


Figure 7: Elapsed average query time for SVPRQs and SVPTQs for the query on the complete video clips.

our new method significantly outperformed the compared methods w.r.t. precision and recall. For $k = 1$, we should retrieve the database object from which the query was generated, we achieved a precision of almost 1.0. For the 2nd nearest neighbor our method still achieved a precision of about 0.9 which is about 40% better than the best of our comparison partners (SMD). The chart displaying the recall of our query results displays a similar picture. The recall of our new methods considerably outperformed the compared methods. Furthermore, we achieved a recall of over 70 % for $k = 3$ which is the average number of similar videos for a query object in our test bed.

The experiments on the queries on parts of video clips display similar results. Our methods outperformed the compared method w.r.t both precision and recall. Though the performance advantage w.r.t. precision was smaller than in the previous experiment, our proposed method still managed to outperform the best comparison partner, SMD, by more than 20% for all values of k. The results w.r.t. recall display similar improvements as well. To conclude, representing video clips as sets of Gaussians is well suited for accurate video retrieval and outperforms method based on sets of feature vectors w.r.t. precision and recall.

For measuring the efficiency of our new methods for query processing, we recorded the time taken for processing all 40 queries representing the complete movie. For each query object, we performed several queries corresponding to several parameter setting ($1 < k < 7$ and $0.1 < P_\Theta < 0.7$). The results are displayed in Figure 7. The average query time for our new methods was approximately 7 times smaller than that of the compared methods. Additionally, it can be seen that using sets of probabilistic feature vectors as query representation did not cause a considerable longer average query time. Let us note that the time for generating the Gaussians of the query was not added to the query time. To conclude our new query algorithm considerably outperformed the compared methods w.r.t. efficiency as well.

6 Conclusions

In this paper, we have proposed efficient techniques for high performance video retrieval. Our methods are based on a summarization technique using probabilistic feature vectors, i.e. Gaussian probability density functions. For storage and efficient retrieval of probabilistic feature vectors, a specialized index structure, the Gauss-tree, has been applied. Every video clip in the database is associated to a set of probabilistic feature vectors. A query video clip is also transformed into either a set of conventional feature vectors or into a set of probabilistic feature vectors. In both cases, query processing involves matching of sets of vectors. We have defined two kinds of set-valued queries, set-valued probabilistic ranking queries and set-valued probabilistic threshold queries, and have proposed efficient algorithms for query evaluation on top of the Gauss-tree. Our experimental evaluation using over 900 music video clips demonstrates the superiority of our approach with respect to both accuracy as well as efficiency of retrieval.

References

[BKK+03] S. Brecheisen, H.-P. Kriegel, P. Kröger, M. Pfeifle, and M. Schubert. Using Sets of Feature Vectors for Similarity Search on Voxelized CAD Objects. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD'03), San Diego, CA, USA*, pages 587–598, 2003.

[BPS06a] C. Böhm, A. Pryakhin, and Matthias Schubert. "Probabilistic Ranking Queries on Gaussians ". In *18th Int. Conf. on Scientific and Statistical Database Management (SSDBM 2006), Vienna, Austria*, 2006.

[BPS06b] C. Böhm, A. Pryakhin, and Matthias Schubert. "The Gauss-Tree: Efficient Object Identification of Probabilistic Feature Vectors". In *22nd Int. Conf. on Data Engineering (ICDE'06)),Atlanta,GA,US*, 2006.

[CKP03] R. Cheng, D.V. Kalashnikov, and Sunil 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.

[CSL99] H. S. Chang, S. Sull, and S. U. Lee. Efficient Video Indexing Scheme for Content-Based Retrieval. In *IEEE Transactions on Circuits and Systems for Video Technology*, volume 9, pages 1269–1279, 1999.

[CXP$^+$04] R. Cheng, Y. Xia, S. Prakhabar, R. Shah, and J.S. 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.

[DYM$^+$05] X. Dai, M. L. Yiu, N. Mamoulis, Y. Tao, and M. Vaitis. ”Probabilistic Spatial Queries on Existentially Uncertain Data”. In *Pro. 9th Int. Symposium on Spatial and Temporal Databases (SSTD2005),Angra dos Reis, Brazil*, pages 400–417, 2005.

[EM97] T. Eiter and H. Mannila. Distance Measures for Point Sets and Their Computation. *Acta Informatica*, 34(2):103–133, 1997.

[GGM02] Hayit Greenspan, Jacob Goldberger, and Arnoldo Mayer. A Probabilistic Framework for Spatio-Temporal Video Representation & Indexing. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV*, pages 461–475, London, UK, 2002. Springer-Verlag.

[Gut84] 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.

[HS95] G.I. 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.

[IBW$^+$04] T. Ianeva, L. Boldareva, T. Westerveld, R. Cornacchia, A. de Vries, and D. Hiemstra. Probabilistic Approaches to Video Retrieval. In *TREKVID*, 2004.

[RB01] J. Ramon and M. Bruynooghe. A polynomial time computable metric between points sets. *Acta Informatica*, 37:765–780, 2001.

[TCX$^+$05] Y. Tao, R. Cheng, X. Xiao, W.K. Ngai, B. Kao, and S. Prabhakar. ”Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions”. In *Proc. 30th Int. Conf. on Very Large Data Bases (VLDB'05), Trondheim, Norway*, pages 922–933, 2005.

[ZRHM98] Yueling Zhuang, Yong Rui, Thomas S. Huang, and Sharad Mehrotra. Adaptive Key Frame Extraction using Unsupervised Clustering. In *ICIP (1)*, pages 866–870, 1998.