Search Support in Data Management Systems

Andreas Henrich

Otto-Friedrich-Universität Bamberg, Germany
andreas.henrich@wiai.uni-bamberg.de

Abstract. In consequence of the change in the nature of data management systems the requirements for search support have shifted. In the early days of data management systems, efficient access techniques and optimization strategies for exact match queries had been the main focus. Most of the problems in this field are satisfactorily solved today and new types of applications for data management systems have turned the focus of current research to content-based similarity queries and queries on distributed databases. The present contribution addresses these two aspects. In the first part, algorithms and data structures supporting similarity queries are presented together with considerations about their integration in data management systems, whereas search techniques for distributed data management systems and especially for peer-to-peer networks are discussed in the second part. Here, techniques for exact match queries and for similarity queries are addressed.

1 Motivation

For decades one main focus of data management systems had been the efficient processing of *fact queries*. A typical—yet rather simple—example would be the search for all open invoices for a customer with a given reference number. Current database management systems are optimized to process large numbers of such queries on dynamic data sets and most problems related to queries of that type are already solved more or less satisfactorily (see [12], for example).

On the other hand, the growing amount of digital documents and digital multimedia documents shifts the requirements for data management systems. In situations where text documents or multimedia documents have to be maintained, a strong requirement for content-based queries is induced. Here a typical query is given by an example document or a query text describing the requested documents. Given a set of images, a query can for example be defined by an example image and in this case query processing is concerned with finding *similar* images. Content-based retrieval has been considered for a long time in the area of information retrieval (see, e.g., [1]). However, information retrieval (IR) has mainly addressed flat text documents in the past. Today, structured

documents (for example, XML documents) and also structured multimedia documents have to be maintained. Many commercial database management systems claim to be the most suitable systems to manage XML data and multimedia data. As a consequence, sophisticated content-based retrieval facilities will be an important distinguishing feature of data management systems.

Another important aspect in data management systems is the decentralized character of cutting edge data management systems. One important trend in this respect are peer-to-peer networks (P2P networks) which are made up of autonomous peers contributing to an administration-free overlay network. The efficient processing of similarity queries in these networks is an important field of research and not yet satisfactorily solved.

In the rest of this contribution, we will first discuss techniques and algorithms for the efficient processing of complex similarity queries in a local scenario. Thereafter, we will discuss approaches towards an efficient processing of content-based similarity queries in P2P networks.

2 Processing Complex Similarity Queries

In recent years, structured multimedia data has become one of the most challenging application areas for data management systems, and search support for structured multimedia data is an important research topic in this respect. To emphasize the key problems in this field, let us assume tree-structured multimedia documents, in which the internal nodes represent intermediate components, such as chapters or sections, and where the leaves represent single media objects such as text, image, video, or audio. In order to support the search for such documents—or document fragments—we need search services which address the following requirements [15]:

- *Dealing with Structured Documents:* The fact that documents are complex-structured objects in our scenario, brings up various interesting research issues.
  First, the search support component of a data management system must allow to search for arbitrary granules ranging from whole documents over intermediate chunks to single media objects. Second, with structured documents many properties of an object are not directly attached to the object itself, but to its components. For example, the text of a chapter will usually be stored in separate text objects associated with the chapter object via links or relationships. Third, additional information about an atomic media object can be found in its *vicinity*. Exploiting the structure of a multimedia document, this concept of vicinity can be addressed navigating one link up and then down to the sibling components.
- *Feature Extraction and Segmentation:* With multimedia data the semantics is usually given implicitly in the media objects. For example, an image might represent a certain mood. Therefore, a search support component should allow to extract features from the media objects potentially describing their semantics. Furthermore, it should

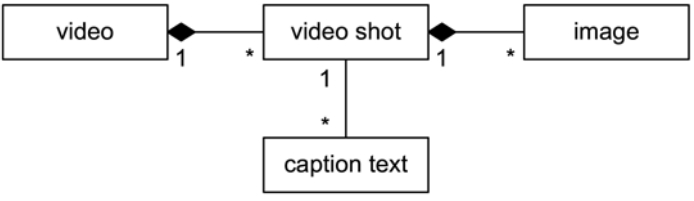

Fig. 1 Small Class Diagram for our Example Query

provide means to subdivide media objects such as images or videos into semantically coherent segments.

- *Similarity Queries:* Because of the vagueness in the interpretation of the media objects and in the formulation of the user's information need, similarity queries should be facilitated. The search support component should provide the definition of ranking criteria for media objects. For example, text objects can be ranked according to their content similarity compared to a given query text. To this end, the well-known vector space model can be applied, for example [1]. Images can be ranked with respect to their color or texture similarity compared to a sample image [34].
- *Combination of Several Similarity Queries:* Images are an example of a media type, for which no single comprehensive similarity criterion exists. Instead, different criteria are applicable addressing, e.g., color, texture, and shape similarity. Hence, algorithms to combine multiple ranking criteria and to derive an overall ranking are needed. To calculate such a combined ranking algorithms such as Nosferatu, Quick-Combine, or *J** have been proposed (cf. section 2.3).
- *Derivation of a Ranking from a Ranking for Related Objects:* With structured documents, ranking criteria are often not defined for the required objects themselves, but for their components or other related objects. An example arises when searching for images where the text in the "vicinity" (e.g. in the same subsection) has to be similar to a given sample text. In such situations, the ranking defined for the related objects has to be transferred to the required result objects. However, neither the semantics nor the implementation of such a "transfer" is self-evident (cf. section 2.4), because there will usually be a *1*:*n*-relationship or even a *n*:*m*-relationship between the objects.

2.1 Components Contributing to Similarity Queries

The main components contributing to the processing of a similarity query can be clarified by an example. Assume a query searching for video shots dealing with hydrogen-powered cars and presenting the cars in a nice sunset scenario. Furthermore, assume that this query has to be processed on a database containing complete videos. Fig. 1 depicts the class diagram underlying our example. This diagram reflects the need to break the videos into shots by means of a shot detection algorithm which in fact is a media-type-

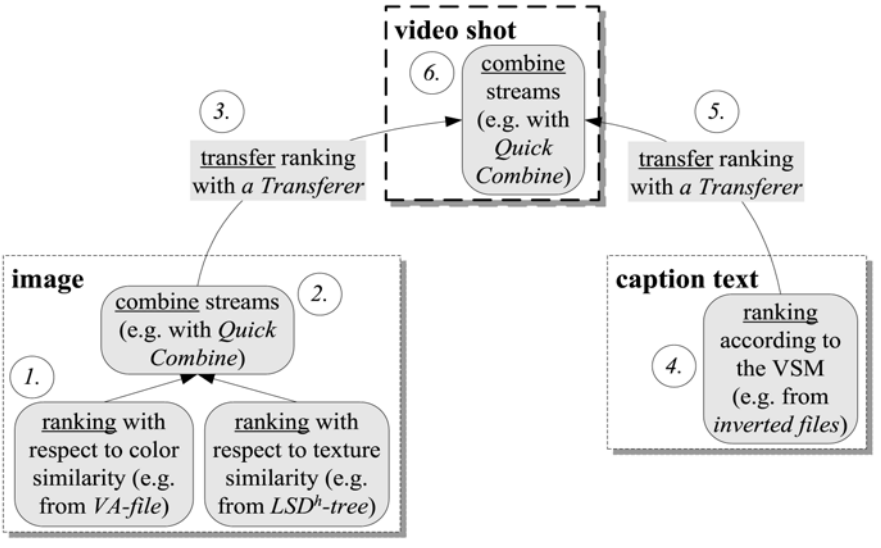

Fig. 2 Steps to Process our Example Query

specific segmentation algorithm. Furthermore, each shot consists of single images, and caption texts are attached to the video shot objects. These caption texts might be created manually or automatically using a speech recognition algorithm. Of course, the segmentation of a video into shots and images as well as the speech recognition could be done at query time; however, it is more reasonable to perform these steps when a video is inserted into the system or by some type of batch process.

When we consider our example query in a little more depth, we note that there are two criteria for a video shot to become a good result of our query:

1. The shot should deal with "hydrogen-powered cars". This criterion can be defined based on the caption text objects associated with the video shot objects and processed using techniques from the field of information retrieval.
2. The shot should present the cars in a "nice sunset scenario". This condition can be tested based on the images constituting the video shots. A video shot containing images similar to an example image representing a sunset might be considered a good result.

It is important to note that both criteria can be interpreted as similarity conditions. With the first criterion, caption texts similar to the query text "hydrogen-powered cars" are required. With the second criterion, images similar to an example image are required. Furthermore, it is important to note that both conditions are not defined for the desired video shot objects directly, but for associated objects.

In the following, we will explain how our example query can be evaluated and which components contribute to the query processing (cf. Fig. 2).

We assume that the segmentation of the videos into shots and images as well as the caption extraction have been performed before query time. Then the query processing consists of the six steps also depicted in Fig. 2:

1. Query processing starts with the "nice sunset scenario" criterion. We assume that access structures for color features and texture features are maintained for the image objects. These access structures (addressed in more detail in section 2.2) both derive a ranking of the image objects according to their similarity to an example image showing a sunset scenario.
2. The first step has yielded two rankings of image objects. Now these two rankings have to be combined into one ranking. In the literature, various algorithms for this purpose have been proposed (cf. section 2.3). The result of this step is a combined ranking of the image objects.
3. Unfortunately, we are not interested in a ranking of image objects, but in a ranking of video shot objects. Therefore, we have to derive a ranking for the related video shot objects from the ranking of the image objects calculated in step 2. To this end, we can exploit that the image objects in the ranking are usually associated with so-called retrieval status values. As a consequence, we can derive a ranking for the video shots associating each video shot with the maximum (or average) retrieval status value of an associated image (cf. section 2.4).
4. Now the second criterion saying that the video shot has to deal with "hydrogen-powered cars" has to be addressed. We start performing a similarity query based on the query text "hydrogen-powered cars" on the caption texts. The usual access structures supporting such text similarity queries are inverted files (cf. section 2.2). This yields a ranking of the caption text objects.
5. Since we are not interested in a ranking of the caption text objects but in a ranking of the video shot objects, we have to transfer this ranking analogously to step 3.
6. Now we have got two rankings for the video shot objects stemming from the "nice sunset scenario" criterion and from the "hydrogen-powered cars" criterion. These two rankings have to be combined into one ranking analogously to step 2.

An important aspect of the process described above is that query processing can be performed in a stream-oriented lazy evaluation approach (client-pull concept). This means that the component computing the final result restricts itself to a small result size in the beginning (maybe the querier is interested in the ten best matches only, with the option for further extensions). Then this component (corresponding to step 6 in Fig. 2) will demand only the minimum amount of input elements from both its input components (corresponding to steps 3 and 5 in Fig. 2) required to calculate the first ten result elements.

In the following we will discuss the single components contributing to the process in more depth. Furthermore, we will discuss query optimization issues in the given context in section 2.5.

2.2 Access Structures for Similarity Queries

Access structures for similarity queries determine a ranking of objects for a set of objects (usually containing all objects of a given type or class), a given query object, and a given similarity measure. Three main categories of access structures can be distinguished: inverted files, trees, and fast sequential scan techniques.

Inverted Files. Inverted files are the most important access structure in information retrieval. Here queries and documents are represented by sparse high dimensional vectors. The dimensionality t of the vectors is given by the number of words occurring in the documents and the vector $(w_{d1},w_{d2},...,w_{dt})$ describing a document d has non-zero entries only for the components corresponding to words occurring in d. Furthermore, the vector $(w_{q1},w_{q2},...,w_{qt})$ representing a query q has non-zero components only for the components corresponding to words occurring in q. In the literature, many approaches have been presented to calculate the concrete values for the non-zero components [1]. Here it is sufficient to know that the components are usually between zero and one.

In this situation, an inverted file stores a list for each component $k \in \{1,...,t\}$ (i.e., for each dimension of the t-dimensional space). And the list for component k contains elements for all documents d with $w_{dk} > 0$. Because the similarity between a query q and a document d is often defined by the scalar product

$$\sum_{k=1}^{t} w_{qk} \cdot w_{dk},$$

we have to access only the lists for components with $w_{qk} > 0$ when processing a query q. The other dimensions cannot contribute anything to the similarity values.

Inverted files are an extremely efficient access structure in situations where two conditions are met: First, the vectors representing the documents should be sparse in order to avoid a huge amount of long lists. Second, the query should contain only few non-zero components, because in such situations only very few of the t lists have to be accessed in order to process the query. If these conditions are not met, inverted files tend to be rather inefficient.

In the literature, various optimization techniques for the implementation of inverted files have been presented (see [4]). Moreover, the use of inverted files has also been suggested in a modified variant for image retrieval [23].

Trees. In the area of spatial data management, many tree-based access structures have been developed starting around 1980. Later on, algorithms for nearest-neighbor queries have been proposed for these structures [13, 19, 31]. When these algorithms were adapted to high-dimensional feature vectors, it turned out that they can yield a sub-linear performance only for dimensions up to about 10.

Fig. 3 depicts a simple tree-based access structure on the left side. The structure is a k-d-tree also used as the basis behind the LSD^h-tree [14] or the hB-tree [22]. The 2-dimensional data space $[0.0; 1.0] \times [0.0; 1.0]$ is partitioned by the root of the tree in dimension 1 at position 0.5. In the left subtree an additional split is made in dimension 2 at position 0.4. The right subtree contains further splits in an analogous way. The corresponding data space partitioning is presented at the right side of Fig. 3. The crosses

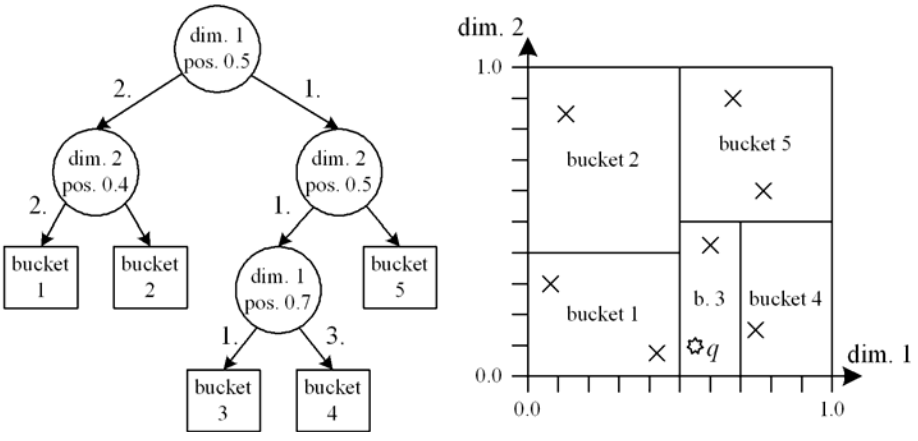

Fig. 3 Example of a Similarity Query on a Tree Structure

represent objects maintained in the access structure. For example, there is an object (0.08;0.3) stored in bucket 1.

If we assume a similarity query with the query vector (0.55; 0.1), represented by a star in Fig. 3, we start processing at the root of the directory and search for the bucket for which the bucket region (i.e., the part of the data space associated with the bucket) contains *q*. Every time during this search when we follow the left son, we insert the right son into an auxiliary data structure *NPQ* (= node priority queue), where the element with the smallest distance between *q* and its data region (i.e., the part of the data space associated with the subtree) has highest priority; and every time we follow the right son, we insert the left son into *NPQ*.

Then the objects in the bucket found (bucket 3 in our example) are inserted into another auxiliary data structure *OPQ* (= object priority queue), where the object with the smallest distance to *q* has highest priority. Thereafter the objects with a distance to *q* less than or equal to the minimal distance between *q* and the data region of the first element in *NPQ* are taken from *OPQ*. Unfortunately, there are no objects with such a small distance in *OPQ* in our example, because the bucket region of bucket 1 is closer to *q*.

Now the directory node or bucket with highest priority is taken from *NPQ*. In our example, this means that we have to follow the path marked with "2." in the tree at the left side of Fig. 3. Bucket 1 determined in this way is processed inserting the objects into *OPQ* and extracting the objects with a distance to *q* less than or equal to the minimal distance between *q* and the first element in *NPQ* from *OPQ*. This way we extract the closest object (0.42; 0.07). If further objects are needed in the ranking, we have to continue the process. In our example we would now follow path "3." in the tree.

The sketched algorithm is rather efficient for spatial applications with 2- or 3-dimensional data and it can also outperform sequential scan techniques for dimensions up to about ten dimensions. However, the *curse of dimensionality* makes it impossible for

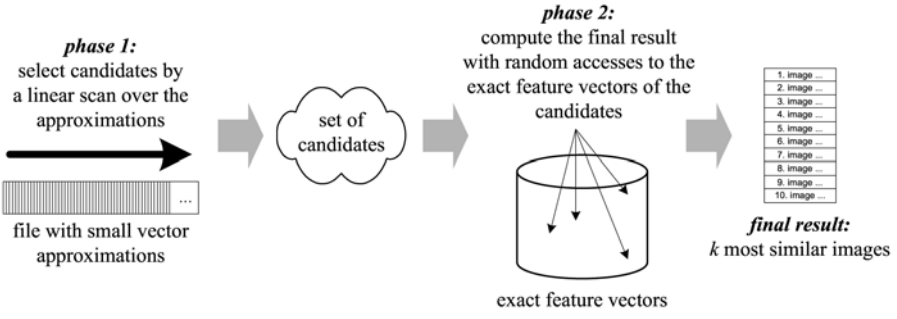

Fig. 4 Similarity Query Searching for the *k* best Matches on a VA-File

tree-based access structures to obtain better than linear complexity for similarity queries in spaces with dimensionality $t > 10$ [38].

Fast Sequential Scan Techniques. The *curse of dimensionality* mentioned above opened the floor for fast sequential scan techniques. As a consequence, the basic idea of the VA-file [38] (*VA* for *vector approximation*) is to accept linear complexity, but to minimize the overhead in order to be faster than tree-based structures. A VA-file stores each feature vector twice. Once in full precision —i.e., with a 32-bit floating point value per component— and once as a rough approximation with about 6 bits per vector component. Of course, the approximations do not suffice to calculate the exact result of a *k*-NN query (searching for the *k* next neighbors or best matches), but a fast linear scan over the approximations can be used to compute a small set of candidates surely including the *k* most similar objects, where *k* can be chosen arbitrarily. Only for the elements in the candidate set the exact feature vectors have to be accessed in order to calculate the *k* most similar objects exactly. Fig. 4 summarizes the phases of such a *k*-NN query.

In the literature, a variety of variants of the VA-file can be found: Balko and Schmidt [3] improve on the VA-file using a different data representation for the approximations and the A-tree proposed by Sakurai et al. [32] improves approximation and query efficiency by creating a tree of approximations. In [26], parallel processing of phase 1 is proposed and, in [25], the IVA-file is proposed as a variant optimized for a specific similarity measure.

None of the three presented types of access structures is superior in all situations. Inverted files are well suited for sparse feature vectors, tree-based structures provide a sublinear performance for low-dimensional feature spaces, and fast sequential scan techniques are well suited for high-dimensional spaces.

2.3 The Data Fusion Problem

In various situations, multiple rankings exist for a set of objects based on different ranking criteria. An example arises when a set of images has to be ranked with respect to the similarity to a given sample image. Here, different similarity criteria addressing color, texture, and also shape are applicable. Hence, components which merge multiple streams representing different rankings over the same base set of objects into a combined ranking are needed.

Because each element of each input stream is associated with some type of similarity value or retrieval status value (RSV), a weighted average over the retrieval status values in the input streams can be used to derive the overall ranking [10]. Other approaches are based on the ranks of the objects with respect to the single criteria—i.e., the position in the stream [15]. To calculate such a combined ranking, efficient algorithms, such as Fagin's algorithm, Nosferatu, Quick-Combine, or *J** have been proposed [9, 29, 11, 27].

The basic idea of the Nosferatu algorithm is to access the input streams in parallel. Let $\alpha_{i,q}$ be the relative importance of input ranking i ($i \in \{1,2\}$ in the case of two input rankings) for query q. Furthermore, let m_i be an upper bound for the $RSV_{i,d}$ values of the elements not yet read from input ranking i. This upper bound can be initialized to some theoretical maximum value at the beginning, and during the algorithm it corresponds to the $RSV_{i,d}$ of the last element read from stream i. The algorithm maintains an auxiliary data structure in which the already calculated part of

$$RSV_{res,d} = \sum_i \alpha_{i,q} \cdot RSV_{i,d}$$

is maintained for each "document" d for which an entry has been read from at least one input stream. If k elements are requested in the result, the algorithm stops as soon as unread entries from the input rankings can no longer affect the top k elements of the output ranking.

In contrast to the Nosferatu algorithm which uses only sequential accesses on the input rankings, the Threshold Algorithm (TA) presented by Fagin is based on sequential and random accesses. This means that as soon as an entry for a document or object d is read from one input stream i, the missing $RSV_{j,d}$ values are determined via random accesses. This means that the algorithm starts with parallel sequential reads from all input rankings. Then the missing $RSV_{j,d}$ values are determined via random accesses. If documents, not yet considered on at least one input stream, can still become part of the top k elements in the result, a further step is performed. Otherwise the process is stopped. Note that

$$RSV_{res,d} \leq \sum_i \alpha_{i,q} \cdot m_{i,d}$$

can be used to calculate the potential retrieval status values of documents not yet considered.

The Quick-Combine algorithm is similar to the Threshold Algorithm. However, it reads only from one list at a time and selects the list for the next sequential read depending on the weights α_i and the current descent in the values of the input rankings. This causes some additional effort for the calculations and potentially fewer accesses on the

input rankings if the assumptions concerning the characteristics of the RSV-values are correct.

Experimental results [17] show that under typical assumptions Quick-Combine will be about 10 to 20% faster than the Threshold Algorithm. Nosferatu performs much worse as long as all α_i values are approximately equal. However, for diverse α_i values Nosferatu can outperform the other algorithms.

2.4 Derived Rankings

As mentioned in the motivation, with structured documents ranking criteria are sometimes not defined for the required objects themselves but for their components or other related objects. An example arises when searching for images where the text nearby (for example in the same section) should be similar to a given sample text. The problem can be described as follows: We are concerned with a query which requires a ranking for objects of some desired object type ot_d (*image* for example). However, the ranking is not defined for the objects of type ot_d, but for related objects of type ot_r (*text* for example).

We assume that the relationship between these objects is well-defined and can be traversed in both directions. In object-relational databases, join indexes and index structures for nested tables are used to speed up the traversal of such relationships. For a further improvement additional path index structures can be maintained on top of the OR-DBMS. Furthermore, we assume there is an input stream yielding a ranking for the objects of type ot_r. For example, this stream can be the output of an access structure supporting similarity queries or a combine algorithm as described in the previous section.

To perform the actual transfer of the ranking, we make use of the fact that each object of type ot_r is associated with some type of retrieval status value (RSV_r) determining the ranking of these objects. As a consequence, we can transfer the ranking to the objects of type ot_d based on these retrieval status values. For example, we can associate the maximum retrieval status value of a related object of type ot_r with each object of type ot_d. Another possibility would be to use the average retrieval status value over all associated objects of type ot_r. The retrieval status value calculated for an object of type ot_d according to the chosen semantics will be called RSV_d in the following.

Based on these assumptions, the "transfer algorithm" [16] can proceed as follows: It uses the stream with the ranked objects of type ot_r as input. For the elements from this stream, the concerned object—or objects—of type ot_d are computed traversing the respective relationships. Then the RSV_d values are calculated for these objects of type ot_d according to the chosen semantics and the object of type ot_d under consideration is inserted into an auxiliary data structure maintaining the objects considered so far. In this data structure, each object is annotated with its RSV_d value. Now the next object of type ot_r from the input stream is considered. If the RSV_r value of this new object is smaller than the RSV_d value of the first element in the auxiliary data structure which has not yet been delivered in the output stream, this first element in the auxiliary data structure can be delivered in the output stream of the transfer component.

Analytical and experimental considerations show that the performance of the transfer of a ranking is heavily influenced by the selected semantics [17]. The maximum semantics yields a better performance than the average semantics. Furthermore, it has to be mentioned that for example the *J** algorithm [27] allows for the integration of a transfer step into the combine algorithm.

2.5 Query Optimization for Similarity Queries

Now that we have sketched some algorithms contributing to the processing of complex similarity queries, it remains to be described how these algorithms can be integrated into the query execution processes of relational databases (see [12], for example).

There are some obvious problems in this respect: First, whether an element belongs to the result of a query or not, cannot be decided based only on the object itself and the query. We have to consider the whole set of objects to find out if there are objects more similar to the query. Second, we are no longer dealing with sets or bags, but with rankings or lists. As a consequence, some standard rules for query optimization are no longer applicable when concerned with similarity or ranking queries.

One important research direction in this respect considers the optimization of so-called *top N* queries. For example, Carey and Kossmann [5] explore execution plans for queries defining a ranking criterion in the `ORDER BY`-clause and a limitation of the result size in a `STOP AFTER`-clause. As another example, Chaudhuri and Gravano [6] study the advantages and limitations of processing a *top N* query by translating it into a single range query that traditional relational DBMSs can process efficiently. In both cases, optimistic and pessimistic variants exist. In the case of the optimistic variants, small intermediate results during query processing are achieved with a risk of ending up with fewer than the desired k best matches. In these cases, a restart of the query processing with larger intermediate results is needed. In short, the range condition used to filter out bad matches has to be relaxed in this case, whereas the pessimistic variants avoid restarts at the price of larger intermediate results.

Another approach presented by Herstel, Schulz, and Schmitt aims for a specific similarity algebra [18, 33]. For this algebra, the applicability of the optimization rules known from classical relational algebra is considered. It turns out that, e.g., weighted combinations of similarity criteria cause problems with respect to associativity and distributivity.

At present, most approaches towards algorithms or systems for complex similarity queries have to be considered as interesting building blocks for specific applications. A general framework with a maturity similar to the frameworks and architectures for processing classical set-oriented relational queries is still an open issue.

3 Processing Similarity Queries in P2P Networks

In section 2, we have described concepts and algorithms for the processing of complex similarity queries in a centralized scenario. We have not considered issues concerning the distribution of the data. However, distributed data management systems and especially the management of data in P2P networks is an important new field of research and data management in grid infrastructures is strongly related. Therefore, we will address concepts and algorithms for similarity queries in P2P networks in this section. We omitted the adjective *complex* here, because currently even the processing of "simple" similarity queries is not really solved for P2P networks.

Roughly spoken, a P2P network is an overlay network where each peer knows its neighbors, which usually form only a small fraction of the P2P network. If a peer wants to distribute a message—maybe a query—in the network, the peer sends the message to its neighbors which, in turn, forward the message to their neighbors and so forth.

To maintain a P2P network three types of administrative operations are needed: (1) *Initial Introduction*: This operation has to be performed when a peer enters the network for the first time. In this case, the peer has to announce itself to the network. (2) *Leave*: When a peer wants to leave the network temporarily there might be a need to do some local reorganization in the network in order to transfer the tasks of the peer to some of its neighbors. (3) *Join*: This operation is used when a peer is entering the network again after a previous leave operation. Then the peer can take over its responsibilities again.

When a peer is in operation, it can perform its intrinsic tasks. For example, it can search for data or information out there in the P2P network. In the following sections, we will mainly concentrate on this operation and discuss *Initial Introduction*, *Leave*, and *Join* only as far as necessary to understand the search operation.

3.1 Looking Up Documents by Their Identifiers

Early P2P systems mainly performed search operations based on *identifiers* rather than content-based similarity queries. A usual query was to look up an item defined by a unique key in the P2P network.

For example, Freenet [7]—whose goal is efficient, anonymity-preserving publication—uses an innovative symmetric lookup strategy for query processing. Within each node a *routing table* is maintained, which contains a list of neighboring peers along with information about which data they furnished in the past. Queries are forwarded from node to node until the desired object is found by a gradient ascent search algorithm with backtracking. In doing so, the goal of anonymity creates some challenges. To provide anonymity, Freenet avoids associating a document with any predictable server or forming a predictable topology among servers. The search is based on unstructured routing tables dynamically built up using caching. As a consequence, unpopular documents may simply disappear from the system, because no server has the responsibility for maintaining replicas. Another consequence is that a search may often need to visit a large fraction of the Freenet network. Freenet is not able to find out in a guaranteed

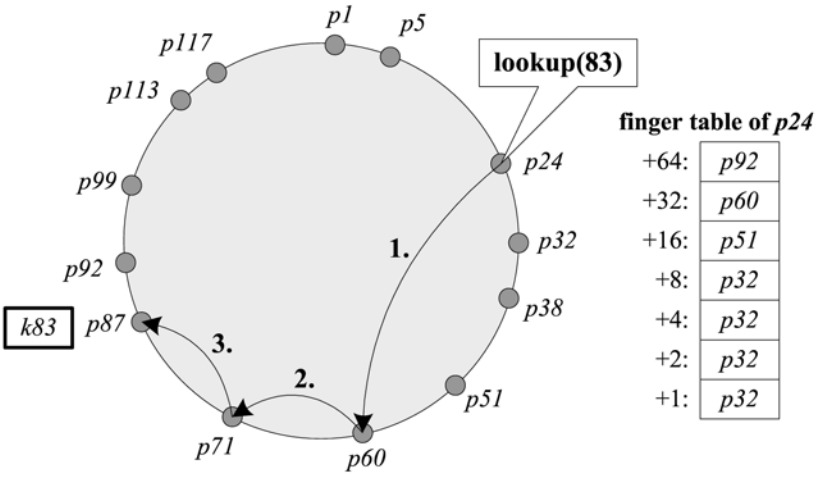

Fig. 5 A Chord Ring and the Course of a Query in the Ring

number of hops, if a given item is present in the P2P network or not. This problem is overcome by distributed hash tables (DHTs) while sacrificing anonymity. In the following, we will consider Chord and CAN as two implementations of a DHT.

Chord — ADHT Organized Like a Skiplist. Chord [36] is an example of a system implementing a DHT (see [2] for an overview with references to various other DHT approaches). With Chord each peer is mapped to a unique ID—usually a 160 bit number. The peers are arranged in a logical ring. Such a ring is given for an ID-length of 7 bit (i.e., IDs from 0 to 127) in Fig. 5. The peers are indicated by small dark grey circles on the ring. On the other hand, objects to be maintained in the P2P network are identified by keys and these keys are also mapped to ID values of the same length. Each object is maintained on the peer with the smallest ID which is equal to or higher than the object's ID. In our example, the object with the ID *k83* is maintained by the peer with ID *p87*.

To speed up the search for an object with a given key, each peer maintains a finger table. This table contains the IP address of a peer halfway around the ring, a quarter around the ring, an eighth around the ring, and so forth. In our example, the finger table for peer *p24* is given. In case of a query, a peer *p* forwards the request to its successor in the ring, except for situations where the ID of the requested object is higher than the ID of a peer in the finger table of *p*. In this case, the request is forwarded to the peer in the finger table with the greatest ID smaller than the ID of the requested object. Fig. 5 depicts the course of a search for the object with ID *k83* issued by peer *p24*. Obviously, the cost of a query (i.e., the number of peers which are involved) grows logarithmically with the number of peers in the network.

To maintain the ring structure in a volatile P2P network, some redundancy has to be introduced [36]. The higher the redundancy, the more reliable the network becomes.

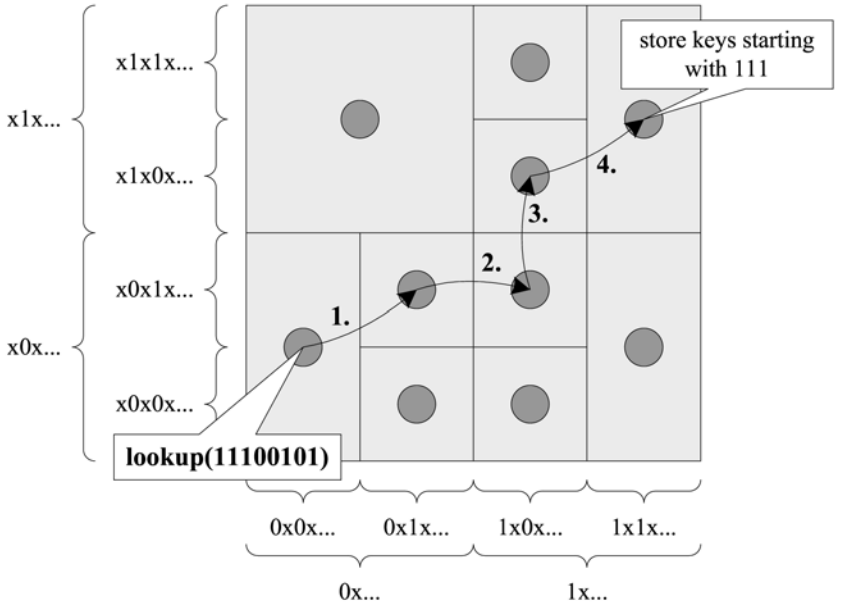

Fig. 6 2-dimensional Coordinate Space of a CAN

Finally, it has to be mentioned that the peer responsible for a given object according to its ID can either maintain the object itself or a reference to the peer physically storing the object.

A Chord-like structure can also be used for simple keyword-based text search operations. Here each object is associated with a small set of keywords. These keywords are mapped to IDs and a reference to an object is maintained on all peers which are responsible for one of its keywords IDs. This can be seen as a first step towards inverted files distributed in a P2P network (cf. section 3.2).

CAN— A *d*-dimensional Implementation of DHTs. Content Addressable Networks (CANs) [30] provide another way to realize a distributed hash table. To this end, a CAN uses a *d*-dimensional Cartesian coordinate space. The coordinate space is partitioned into hyper-rectangles, called *zones*. Each peer is responsible for a zone. A key is associated with a point in the coordinate space. For example, a key can be mapped to an ID and the ID can be used to generate the coordinate values by interleaving. For $d = 2$, the first, the third, the fifth, … bit form the coordinate value in the first dimension and the second, the forth, the sixth, … bit form the coordinate value in the second dimension. This way each object is maintained at the peer whose zone contains the coordinates derived from the object's key. In order to process queries, each peer maintains a routing table of all its bordering neighbors in the coordinate space (cf. Fig. 6).

If we search for an object with a given key, the query is forwarded along a path that approximates the straight line in the coordinate space from the peer posing the query to the peer maintaining the key. To this end, each peer forwards the query to its neighbor closest in the coordinate space to the peer maintaining the key. The cost of a query processed this way is $O(d \cdot \sqrt[d]{N})$ where N denotes the number of peers in the network. This becomes $O(\sqrt{N})$ for $d = 2$.

At first glance, one might think that a CAN could also be used for similarity queries in a d-dimensional space. For this, we could use d-dimensional feature vectors instead of interleaved IDs in order to position the objects in the d-dimensional coordinate space. Unfortunately, this approach fails because, due to the curse of dimensionality, we would have to access all peers in order to obtain the most similar object in this way for $d > 10$. Furthermore, the administrative data which has to be maintained on each peer would be rather huge for high values of d. Another weak aspect from our point of view is that CANs (as any distributed index structure) require sending a substantial amount of feature vectors (possibly all of them) to other peers when entering the network ([37] takes this approach).

3.2 Performing Similarity Queries in P2P Networks

While most DHTs deal with one-dimensional IDs (CANs, as an exception use low-dimensional vector spaces for routing), there is now more and more research concerning information retrieval queries in P2P networks.

Performing Content-Based Queries on Text Data. Joseph [20] proposes NeuroGrid, which (like some other current P2P approaches) permits keyword-based search for annotated items. NeuroGrid exploits the fact that each item is described by just a few keywords. As in Freenet, routing tables are maintained in each peer to guide the query processing.

FASD [21] uses the same query routing scheme as Freenet, however, it uses *term vectors* of full documents (*i.e.,* the vector of words present/absent in a document) to make the routing decisions. This approach is particularly effective in situations where the queries consist of only a few keywords. On multi-keyword queries, however, this scheme is likely to suffer from the curse of dimensionality.

Cuenca-Acuna and Nguyen [8] provide full-text information retrieval in their PlanetP system. In PlanetP, the content of each peer is summarized using a *bloom filter*. A bloom filter is a bit array that represents a set S with fewer than $|S|$ bits. In this case, the set represented by the bloom filter is the *set of terms* (words) present in a peer. Each peer distributes its bloom filter using a rumor-spreading algorithm. Thus after a short period of time *all peers* contain the descriptors of *all other peers* within the network. Query processing is done by visiting peers starting with the most probable provider of a match.

Similarity Queries for Multimedia Data. Tang et al. [37] present two approaches for text retrieval in P2P systems and suggest that these approaches can be extended

to image retrieval. One of them (pVSM, peer vector space model) stores *(term, document_id)* pairs in a DHT, and then retrieves such pairs to perform text retrieval. According to the authors this approach can be handled due to the fact that terms are Zipf distributed, and one can thus limit the index for each document to a few strongly weighted terms. This approach seems to be hard to translate to image data: [35] reports the need for a large number of features when using text information retrieval methods on image data.

The other approach suggested in [37], pLSI, is based on Latent Semantic Indexing: Singular Value Decomposition is used to decrease the dimensionality of the feature vectors, and the reduced feature vectors are stored in a hierarchical version of a CAN (eCAN). For each *document* of the collection, *(feature vector, document id)* pairs are stored in the eCAN. The curse of dimensionality is addressed by partitioning each feature vector into several lower-dimensional feature vectors. Each of these vectors is said to be in a *plane*. The first of these lower-dimensional partial vectors is stored in a CAN_1, the second in CAN_2 and so on through CAN_m. A given query is then split into several queries of lower dimensionality. Together with aggressive pruning, Tang et al. manage to achieve impressive results on text data: few nodes need to be accessed for achieving a high precision. The future will have to show whether or not these results can also be achieved on image data.

Ng and Sia [28] present the Firework Query Model for information retrieval and CBIR in P2P networks. In the Firework Query Model, there are two classes of links, normal *random* links, and privileged *attractive* links. A query starts off as a flooding query forwarded to all neighbors. If a peer deems the query too far away from the peer's local cluster centroid, it will forward the query via a random link, decreasing the time to live (TTL) of the query, which gives a limit for the number of hops a query can make. Otherwise, it will process the query, and forward it via all its attractive links without decreasing the TTL. From these approaches two main research directions can be identified:

- pVSM can be seen as a distributed implementation of an access structure—or, more specifically, of inverted files. Here the single files carrying references to the objects/documents containing a certain word are distributed in the P2P network. For a query consisting only of very few words this might be adequate. However, with queries containing multiple terms the merging of the lists will cause a significant communication overhead.
- Techniques like PlanetP are based on short content descriptions for each peer's content. In PlanetP, each peer stores compact descriptions of *all* other peers in the network. With a given query, it can forward the query only to those peers which seem promising according to their compact description.

A Scalable Approach to Similarity Queries Using Peer Descriptions. To conclude the considerations on similarity queries in P2P networks, we want to sketch the Rumorama approach [24] aiming towards scalable summary-based search in P2P networks. The goal of the Rumorama protocol is to establish a robust hierarchy of Plan-

etP networks. The leaves within the Rumorama hierarchy are PlanetP networks (called leaf nets). Inner nodes within the Rumorama hierarchy allow accessing the leaf nets.

The situation in a Rumorama net is depicted in Fig. 7 analogously to the situation for a CAN in Fig. 6. As usual, each peer is identified by an ID. The ID of the peer used as the querier in Fig. 7 is 01011000... . This peer is located in a leaf net (indicated by a grey rectangle with a dashed border) and it maintains compact peer descriptions for all 18 peers in that leaf net. With all peers in the leaf net the peer shares the first two bits "01" of its ID. More precisely, each peer maintains a so-called *friends' mask* representing the prefix of the ID which is equal for all peers in the leaf net. In our case, the friends' mask is 01. In order to communicate with peers in other leaf nets, our peer maintains additional information about simple neighbors. In contrast to friends in the leaf net, the peer does not maintain compact peer descriptions for its neighbors. Our example peer has two neighbors for each bit in its friends' mask. It has a neighbor with an ID starting with 0 and it has a neighbor with an ID starting with 1 for the first bit. For the second bit, it has a neighbor with an ID starting with 00 and it has a neighbor with an ID starting with 01.[1]

When a peer wants to issue a query it is important to distribute the query to all leaf nets. For this, the querier forwards the query to its neighbors with an ID starting with 0 and with an ID starting with 1. In addition, the query is augmented with a parameter denoting the length of the prefix already covered in the distribution process (1 for the first step). The two peers, contacted in that way, check whether the length of the prefix *pre* already covered in the distribution process is equal to or greater than the length of their friends' mask. In that case, a leaf net is reached and in this leaf net the query can be processed for all peers with prefix *pre*. Otherwise, the query is forwarded to the neighbors maintained by these peers at the next level.

In Fig. 7, the course of a query is sketched. The indicated querier forwards the query to two neighbors (one with an ID starting with 0 and one with an ID starting with 1). For these two peers, leaf nets are not yet reached and therefore the query is again forwarded to the respective neighbors for prefix length two.

The prefixes are indicated in Fig. 7 in squared brackets. Note that this second step is processed in parallel on the two peers reached in the first step. For the peers reached with the prefixes 11, 10 and 01 leaf nets are now reached. Only the peer reached for the prefix 00 has to forward the query one more time. Now five leaf nets forming a disjoint and exhaustive partitioning of the whole P2P network are reached and the query is processed in each leaf net. If the querier is interested in the k best matches, they are calculated for each leaf net separately. Thereafter the results obtained for the leaf nets are merged following the paths on which the query was distributed in reverse order.

Obviously, this schema cannot overcome the curse of dimensionality. But it allows for a scalable implementation of PlanetP-like P2P networks. Peers have to maintain

1. Note that it is necessary in the concrete implementation to maintain some redundant neighbors in order to keep the system fit for work in a volatile network. Furthermore, it has to be noted that each peer can decide the length of its friends' mask independently and that this length can be changed during the lifetime of a peer.

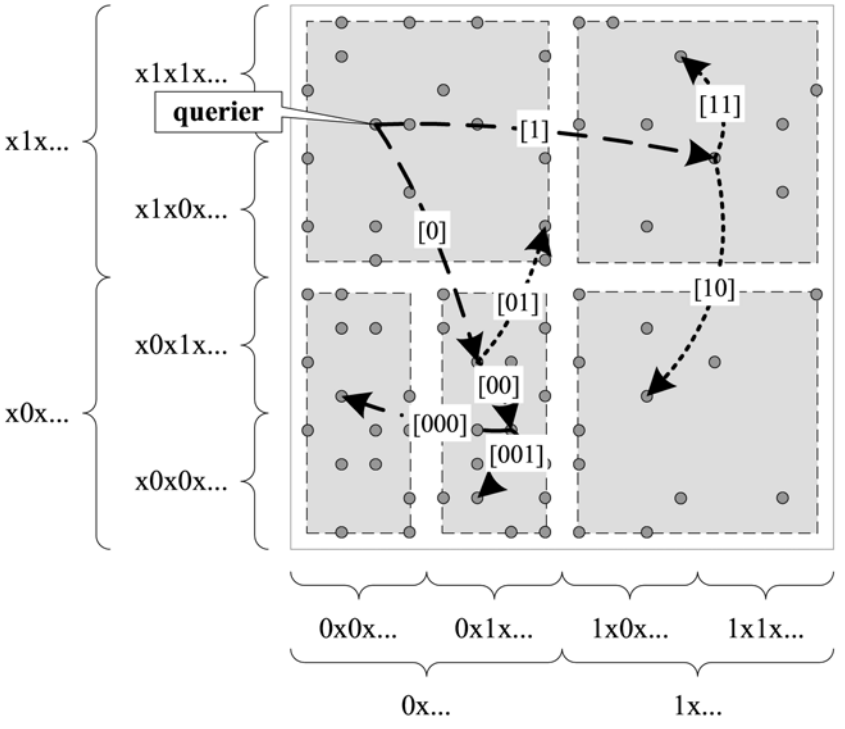

Fig. 7 Distribution Example of a Similarity Query to the Leaf Nets in Rumorama

compact peer descriptions only for a small set of peers with tunable size and the query distribution can be performed in logarithmic time. For the local processing of the queries in the leaf nets, various heuristics trading result quality for processing time are applicable. For example, we could access only a fixed fraction of the peers in each leaf net concentrating on the peers most promising according to their compact representation.

4 Conclusion

In the present contribution, we have discussed the efficient processing of similarity queries in a central scenario and in a P2P scenario. Since algorithms for simple similarity queries have reached a mature level for the central scenario the focus is now turning to complex similarity queries with multiple ranking criteria based on a complex object schemata. For the P2P scenario, scalable algorithms for content-based similarity queries are currently coming up. Here thorough analytical and experimental considerations of the upcoming concepts will be necessary in order to reach a mature and commercially applicable state of the systems.

References

[1] Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*. ACM press. Pearson Education Limited, Harlow, England, 1999.

[2] Balakrishnan, H., Kasshoek, M. F., Karger, D., Morris, R., Stoica, I.: Looking Up Data in P2P Systems. *Commun. ACM*, 46(2):43–48, Feb. 2003.

[3] Balko. S., Schmitt, I.: Efficient Nearest Neighbor Retrieval by Using a Local Approximation Technique - the Active Vertice Approach. Technical Report 2, Fakultät für Informatik, Universität Magdeburg, 2002.

[4] Buckley, C., Lewit, A.: Optimization of inverted vector searches. In *Proc. 8th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 97–105, New York, USA, 1985.

[5] Carey, M. J., Kossmann, D.: On saying "Enough already!" in SQL. In *Proc. 1997 ACM SIGMOD Intl. Conf. on Management of Data*, pages 219–230, Tucson, Arizona, 13–15 June 1997.

[6] Chaudhuri, S., Gravano, L.: Evaluating top-*k* selection queries. In *Proc. 25th Intl. Conf. on Very Large Data Bases, September, 1999, Edinburgh, Scotland, UK*, pages 397–410, 1999.

[7] Clarke, I., Sandberg, O., Wiley, B., Hong, T. W.: Freenet: A distributed anonymous information storage and retrieval system. In H. Federrath, editor, *Designing Privacy Enhancing Technologies*, volume 2009 of *LNCS*, 2000.

[8] Cuenca-Acuna, F. M., Nguyen, T. D.: Text-Based Content Search and Retrieval in ad hoc P2P Communities. Technical Report DCS-TR-483, Department of Computer Science, Rutgers University, 2002.

[9] Fagin, R., Lotem, A., Naor, M.:. Optimal aggregation algorithms for middleware. In *Proc. 10th ACM Symposium on Principles of Database Systems: PODS*, pages 102–113, New York, USA, 2001.

[10] Fagin, R., Wimmers, E. L.: A formula for incorporating weights into scoring rules. *Theoretical Computer Science*, 239(2):309–338, May 2000.

[11] Güntzer, U., Balke, W.-T., Kießling, W.: Optimizing multi-feature queries for image databases. In *VLDB 2000, Proc. 26th Intl. Conf. on Very Large Data Bases*, pages 419–428, Cairo, Egypt, 2000.

[12] Härder, T., Rahm, E.: *Datenbanksysteme: Konzepte und Techniken der Implementierung*. Springer, Heidelberg, 2nd edition, 2001.

[13] Henrich, A.: A distance scan algorithm for spatial access structures. In *Proc. of the 2nd ACM Workshop on Advances in Geographic Information Systems*, pages 136–143, Gaithersburg, Maryland, USA, 1994. ACM Press.

[14] Henrich, A.: The LSD^h-tree: An access structure for feature vectors. In *Proc. 14th Intl. Conf. on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*, pages 362–369. IEEE Computer Society, 1998.

[15] Henrich, A., Robbert, G.: Combining multimedia retrieval and text retrieval to search structured documents in digital libraries. In *Proc. 1st DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries*, pages 35–40, Zürich, Switzerland, Dec. 2000.

[16] Henrich, A., Robbert, G.: Ein Ansatz zur Übertragung von Rangordnungen bei der Suche auf strukturierten Daten. In *Tagungsband der 10. Konferenz Datenbanksysteme für Business, Technologie und Web (BTW 2003)*, volume 26 of *LNI*, pages 167–186, Leipzig, Deutschland, Feb. 2003. GI.

[17] Henrich, A., Robbert, G.: Comparison and evaluation of fusion algorithms and transfer semantics for structured multimedia data. In C. Danilowicz, editor, *Multimedia and Network Information Systems (Vol. 2) (1st International Workshop on Multimedia Information Systems Technology)*, pages 181–192, Szklarska Poreba, Poland, September 2004. Oficyna Wydawinicza Politechniki Wroclawskiej.

[18] Herstel T., Schmitt, I.: Optimierung von Ausdrücken der Ähnlichkeitsalgebra SA. In *INFORMATIK 2004 - Informatik verbindet - Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), Band 2*, volume P-51 of *LNI*, pages 49–53, Ulm, Germany, September 2004.

[19] Hjaltason, G. R., Samet, H.: Ranking in spatial databases. In *Advances in Spatial Databases, 4th International Symposium, SSD'95, Portland, Maine, USA, August 6-9, 1995, Proceedings*, volume 951 of *LNCS*, pages 83–95. Springer, 1995.

[20] Joseph, S.: Adaptive routing in distributed decentralized systems: Neurogrid, Gnutella and Freenet. In *Proc. of workshop on Infrastructure for Agents, MAS, and Scalable MAS, at Autonomous Agents*, Montreal, Canada, 2001.

[21] Kronfol, A. Z.: A Fault-tolerant, Adaptive, Scalable, Distributed Search Engine, May 2002. Final Thesis, Princeton.
URL: http://www.searchlore.org/library/kronfol_final_thesis.pdf

[22] Lomet, D. B., Salzberg, B.: The hb-tree: A multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, 1990.

[23] Müller, H., Squire, D. M., Müller, W., Pun, T.: Efficient access methods for content-based image retrieval with inverted files. Technical Report 99.02, Computer Vision Group, University of Geneva, July 1999.

[24] Müller, W., Eisenhardt, M., Henrich, A.: Scalable summary-based search in P2P networks. submitted for publication, 2004.

[25] Müller, W., Henrich, A.: Faster exact histogram intersection on large data collections using inverted VA-files. In *Image and Video Retrieval: 3rd Intl. Conf.* CIVR. Proceedings, volume 3115 of LNCS, pages 455-463, Dublin, Ireland, July, 2004. Springer.

[26] Müller, W., Henrich, A.: Reducing I/O cost of similarity queries by processing several at a time. In Proc. MDDE '04, 4th International Workshop on Multimedia Data and Document Engineering, Washington DC, USA, July 2004. IEEE Computer Society.

[27] Natsev, A., Chang, Y.-C., Smith, J. R., Li, C.-S., Vitter, J. S.: Supporting incremental join queries on ranked inputs. In VLDB 2001, Proc. of 27th Intl. Conf. on Very Large Data Bases, pages 281–290, Roma, Italy, 9 2001.

[28] Ng, C. H., Sia, K. C.: Peer clustering and firework query model. In Poster Proc. of The 11th Interational World Wide Web Conf., Honululu, HI, USA, May 2002.

[29] Pfeifer, U., Pennekamp, S.: Incremental Processing of Vague Queries in Interactive Retrieval Systems. In Hypertext - Information Retrieval - Multimedia '97: Theorien, Modelle und Implementierungen integrierter elektronischer Informationssysteme, pages 223–235, Dortmund, 1997. Universitätsverlag Konstanz.

[30] Ratnasamy, S., Francis, P., Handley, M., Karp, R., Schenker, S.: A scalable content-addressable network. In Proc. 2001 Conf. on applications, technologies, architectures, and protocols for computer communications, San Diego, CA, 2001.

[31] Roussopoulos, N., Kelley, S., Vincent, F.: Nearest neighbor queries. In Proc. 1995 ACM SIGMOD Intl. Conf. on Management of Data, San Jose, California, May 22-25, 1995, pages 71–79, 1995.

[32] Sakurai, Y., Yoshikawa, M., Uemura, S., Kojima, H.: The A-tree: An index structure for high-dimensional spaces using relative approximation. In Proc. of the 26th Intl. Conf. on Very Large Data Bases, pages 516–526, Cairo, 2000.

[33] Schmitt, I., Schulz, N.: Similarity relational calculus and its reduction to a similarity algebra. In 3rd Intl. Symposium on Foundations of Information and Knowledge Systems (FoIKS'04), volume 2942 of LNCS, pages 252–272, Austria, 2004. Springer.

[34] Smith, I., Chang, S.-F.: VisualSEEK: A fully automated content-based image query system. In Proc. of the 4th ACM Multimedia Conf., pages 87–98, New York, USA, Nov. 1996.

[35] Squire, D. M., Müller, W., Müller, H., and Raki, J.: Content-based query of image databases, inspirations from text retrieval: inverted files, frequency-based weights and relevance feedback. In 11th Scandinavian Conf. on Image Analysis, Kangerlussuaq, Greenland, 1999.

[36] Stoica, I., Morris, R., Karger, D., Kaashoek, F., Balakrishnan, H.: Chord: A scalable Peer-To-Peer lookup service for Internet applications. In Proc. ACM SIGCOMM Conf., San Diego, CA, USA, 2001.

[37] Tang C., Xu, Z., Mahalingam, M.: pSearch: Information retrieval in structured overlays. In First Workshop on Hot Topics in Networks (HotNets-I), Princeton, NJ, 2002.

[38] Weber, R., Schek, H.-J., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In Proc. Intl. Conf. on VLDB, New York, USA, 1998.