Fast Evaluation Techniques for Complex Similarity Queries

Klemens Böhm Michael Mlivoncic Hans-Jörg Schek Roger Weber

Database Research Group, Swiss Federal Institute of Technology Zurich, Switzerland
{boehm, mlivoncic, schek, weber}@inf.ethz.ch

Abstract

Complex similarity queries, i.e., multi-feature multi-object queries, are needed to express the information need of a user against a large multimedia repository. Even if a user initially issues a single-object query over one feature, a system with relevance feedback will automatically generate a complex similarity query. Relevance feedback is only useful if response times are interactive. Therefore, this article contributes to the important problem how to evaluate such complex queries efficiently. We describe a new evaluation technique called *Generalized VA-File-based Search (GeVAS)*. It builds on the VA-File [27], supports queries over several feature types, and borrows the idea to search an index structure with several query objects in parallel from Ciaccia et al. [8]. Our main contributions are twofold: 1) we show that GeVAS does not degenerate for queries with many objects or many feature types. 2) We develop a number of variants of GeVAS, tailored to the different distance measures and distance-combining functions, and we show that they yield a significant performance improvement.

1 Introduction

Similarity search has been touted as an effective approach to find relevant images in a multimedia document collection. In the conventional case, the user provides a reference image, and the infrastructure identifies the images that are most similar. However, such a query image typically is only a rough approximation of what the user is seeking. Typically, one wants to bring together characteristics of various reference images, and to have different similarity measures for different information needs. Thus, we need complex similarity queries to express a given information need. In what follows, a complex similarity query (or complex query, for short) is a *multi-object multi-feature query*, i.e., a query with several query images and explicit references to different feature types.

An important application for such queries is *relevance feedback*, a concept that increases the result quality of similarity search. The idea behind relevance-feedback is that the system interprets user judgements on preliminary search results to generate a new, more precise query. While early relevance-feedback models, notably [21], generate a new, artificial query point, more recent models generate queries with several query objects (see [12] for *query expansion* with text documents), or they prune, add or reweight feature types [22]. Preliminary experiments have shown that different models are most adequate for different scenarios [22, 18]. Thus, relevance feedback requires complex queries. Furthermore, to be acceptable to the user, a system featuring relevance feedback must evaluate such queries efficiently over large image collections. Previous work on relevance feedback over images was restricted to small collections, e.g., [22].

Briefly, the objective of this article is to develop fast evaluation techniques for complex similarity queries that enable relevance feedback methods for very large collections. Existing approaches on complex query evaluation either are not general enough [8], or there typically is a large gap between the evaluation times of simple queries and complex ones. This is the case for the well-known A_0-algorithm [9] and a recent elegant extension called Quick-Combine [11]. Furthermore, their applicability is restricted to components that implement the generic interface required [28].

Subsequently, we assume that we are free to choose the physical organization of the data and the search algorithms on top. This article proposes the GeVAS set of algorithms ('GeVAS' = 'Generalized VA-File based Search'). GeVAS generalizes and combines existing techniques in a natural way, namely the *VA-File* [27], the *CPZ-approach* [8], as we call it, and *parallel access to indices*, a concept subsequently denoted as *feature fusion on-the-fly*. On the other hand, its investigation in quantitative terms is an open issue. The contributions of this article are as follows:

1. **Description of GeVAS.** While the approach is natural, there are some problems regarding implementation that we will address.

**Proceedings of the 27th VLDB Conference,
Roma, Italy, 2001**

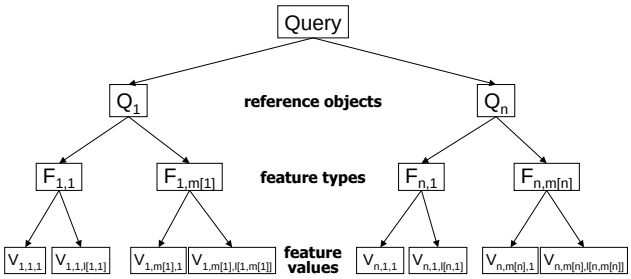


Figure 1: Query Model

2. **Providing evidence that GeVAS is feasible.** VA-File based similarity search computes bounds on the query distance of each data object. If these bounds are not tight, the technique would degenerate. We show that this is not the case by a formal analysis and by extensive experiments covering all relevant cases.
3. **Refinement of GeVAS for different similarity measures.** We investigate the relationship between the similarity measure in use and query-evaluation times. These variants are either not obvious, or the benefit in quantitative terms is not obvious.
4. **Extensive evaluation of the GeVAS variants.** We compare the different variants of GeVAS experimentally, using large image collections. We achieve significantly better performance in most cases: For instance, one optimization (*precomputation of distances*) has the effect that query-evaluation time is independent from the number of data objects

The remainder of this article has the following structure: Section 2 introduces our model of complex similarity queries. Section 3 describes the evaluation alternatives. Section 4 summarizes our formal analysis of the feasibility of GeVAS. Section 5 is a description of our experimental setup. Section 6 presents our experimental results. Section 7 concludes. [5] is an extended version of this article. A prototype system which implements all these concepts is available on the Web [15].

2 Query Structure and Semantics

A common implementation of image similarity search is as follows: a number of *feature-extraction algorithms* map each image to points in *feature spaces*. Similar images are mapped to points that lie close to each other in the feature spaces. Hence, an evaluation of a *simple similarity query*, i.e., a query with one reference image and one feature, is implemented as *nearest-neighbor search (NN-search)* in the respective feature space.

2.1 Query Structure

This subsection describes the structure of complex queries which is borrowed from [19, 6]. Figure 1 displays the general structure of a query. A query consists of *n reference objects* $\Gamma = \{Q_1, \ldots, Q_n\}$. For reference object, there may be an arbitrary number of feature types ($F_{i,1}, \ldots, F_{i,m[i]}$, for each Q_i). Finally, the point $(v_{i,j,1}, \ldots, v_{i,j,l[i,j]})$ is the representation of query object Q_i in feature space $F_{i,j}$. For the sake of presentation, Figure 1 omits the weights at the various levels of the query model. The following examples provide some intuition regarding the structure of a query.

Example 1 (Complex Query) *A user is looking for an image of a black cow, but he only has a picture of a brown cow and one of a black horse at hand. The corresponding query has two reference objects Q_1 (cow) and Q_2 (horse). There is one feature type for each image: $F_{1,1}$ is the shape feature of Q_1, $F_{2,1}$ the color feature of Q_2.*

Example 2 (Relevance Feedback) *Usually, a user would not type in a complex query as insinuated in Example 1. Rather, he tells the system which images he likes and for what reasons. To continue the above example, his initial query might have returned an image A with a black horse and image B with a brown cow. The user would rate the images as follows: image A contains an object with the* right color *but* wrong shape, *and image B has the* right shape *but the* wrong color. *The resulting query is equal to the one in Example 1 but is now generated by the system. Note that relevance feedback based on Rocchio [21] will often not suffice. With the example, Rocchio would produce artificial vectors for the shape and color feature that lie between the two vectors for the images.*

2.2 Query Semantics

Distances. To state what a similarity query stands for, we introduce the notions of *feature distance*, *reference-object distance*, and *query distance*. The feature distance $x_{i,j}$ measures the dissimilarity of reference object Q_i and data object P regarding feature type $F_{i,j}$. The i-th reference-object distance x_i of a data object is a *combined distance* to the *i*-th reference object with regard to the feature representations $F_{i,1}, \ldots, F_{i,m[i]}$. We abbreviate this distance of data object P with $\delta_i(P)$. Usually, there are several ways to combine the distances of features to an overall distance, e.g., Weighted Average. Moreover, we allow to weight the individual features to reflect their importance for the current information need. Finally, the query distance is a combined distance over all reference objects. We also allow for weighting on this level of the query model. In the following, we use the abbreviation $\delta(P)$ for the query distance of P.

Distance-Combining Functions. Each object P_a of the database D and each reference object Q_i is mapped to a set of feature representations $\{\mathbf{p}_{a,1}, \ldots, \mathbf{p}_{a,m}\}$ and $\{\mathbf{q}_{i,1}, \ldots, \mathbf{q}_{i,m}\}$, respectively. The l-th feature representation of an object lies in a d_l-dimensional feature space. A distance measure $\delta_l(\mathbf{p}_{a,l}, \mathbf{q}_{i,l})$ defines the distance of two points in this space. A common measure is the (weighted) Minkowski metric L_s:

$$L_s: \ \delta_l(\mathbf{p}_{a,l}, \mathbf{q}_{i,l}) \ = \ \sqrt[s]{\sum_{j=1}^{d_l} w_j \cdot |(p_{a,l})_j - (q_{i,l})_j|^s} \qquad (2.1)$$

where $(p_{a,l})_j$ is the j-th component $\mathbf{p}_{a,l}$. Throughout this paper, we may omit l or s, as well as the weights, when the explicit mention is not needed. Based on the L_2 metric, we introduce a distance function $(L_2)^2$ that avoids the square root. Although this distance function is not a metric, it yields the same ordering of data points as the L_2-metric. But its computation is significantly cheaper.

To describe the reference-object distance and the query distance, we introduce the notion of *distance-combining function*. It combines a set of distances to an overall distance. More formally:

Definition 2.1 (Distance-Combining Function) *$\Delta : (\mathbb{R})^m \to \mathbb{R}$ is a distance-combining function for m distances. Δ is monotonic in all arguments, i.e. $\Delta(x_1,\ldots,x_m) \leq \Delta(x'_1,\ldots,x'_m)$ if $x_i \leq x'_i$ for all i.*

Note that monotonicity is a natural requirement: if one of the distances becomes larger, the similarity decreases. The following distance-combining functions implement different ways of combining distances. The examples combine reference-object distances to a query distance.

Example 3 (Fuzzy-And, Fuzzy-Or, Weighted Average) *Given a set of n reference objects,* Fuzzy-And *implements the idea that a data object should be close to all of them. This leads to the maximum function (*max*) for Δ, i.e.: $\delta(P) = \Delta_{and}\big(\delta_1(P),\ldots,\delta_n(P)\big) = \max_{i=1}^{n} \delta_i(P)$.*

If the reference objects represent a set of possibilities, and each of them is acceptable, the Fuzzy-Or *distance-combining function would be more appropriate. Rather than $\Delta = \max$, it uses $\Delta = \min$.*

Finally, let $\mathbf{w} \in \mathbb{R}^n$ be the weights of the individual distances such that $\sum_{i=1}^{n} w_i = 1$. Then Weighted Average *is given as $\Delta = \sum_{i=1}^{n} w_i \cdot$.*

Normalization and Weighting. Combining feature distances as described so far is not sufficient since the meaning of a distance value depends on the distance distribution. For example, assume we would deploy Fuzzy-Or on a distance from the feature space $[0,1]^{10}$ and on one from the feature space $[-10,10]^{100}$. As distances in the first space are much smaller than in the later one, Fuzzy-Or yields the distance from the first space in almost all cases. This would mean that the second feature did not matter with regard to overall similarity. We normalize distances with a well-chosen normalization function for each distance measure to avoid such situations. A common normalization scheme is a linear transformation using the Gaussian parameters of the distance distribution of δ:

Example 4 *Let μ_l and σ_l be the mean distance and the standard deviation of distances in the l-th feature space using the distance function δ_l. Then the Gaussian normalized distances $\hbar_{\delta_l}(x)$ are given by: $\hbar_{\delta_l}(x) = \frac{x-\mu_l}{\sigma_l}$.*

Note that normalized distances may be smaller than 0. As such, the term "distance" is not appropriate as distances are elements of $\mathbb{R}^+$. But for the sake of clarity, we do not introduce a new name.

Some relevance-feedback models further require weighting of individual reference objects, feature types and dimensions within a single feature vector. Dimension weights are implemented by the distance function, e.g., w_i in Equation 2.1. Weighting reference objects and feature types is straightforward with Weighted Average. But this is not the case with Fuzzy-And or Fuzzy-Or. Fagin et al. solved this problem for a slightly different setting, and we deploy their solution in our context (cf. [10]).

Problem Statement. Finally, we can define the nearest neighbor search problem for complex queries.

Definition 2.2 (Nearest Neighbor Search; NN-Search) *Let $\delta(P)$ be the query-distance function for the data object P, given a set of reference objects and a family of features, as described by the query model. Further, let k be the number of nearest neighbors to return. The answer set A of a k nearest neighbor search (k-NN-search) is:*

$$A = \Big\{ P \;\Big|\; P \in D \wedge \mathit{rank}(P) \leq k \Big\} \qquad \mathit{with}$$

$$\mathit{rank}(P) \;=\; 1 + \sum_{P' \in D} \begin{cases} 1 & \delta(P') < \delta(P) \\ 0 & \mathit{otherwise} \end{cases}$$

With the above definition, the size of the answer set A may be larger than k, e.g., if several objects P have $\mathit{rank}(P) = k$. If the user insists on seeing only k answers, we may drop the $|A| - k$ objects with the largest rank from A in a non-deterministic way.

While distances measure dissimilarity, it would also be possible to work with *scores* that measure the similarity. Scores are *normalized* by definition, i.e., from the interval $[0,1]$ (1=identity, 0=no similarity at all). We deploy a *correspondence function* [8] to map query distances to overall scores. Hence, unlike [9, 8, 28], we do not need to deal with scores at the intermediate levels.

3 Evaluation of Complex Queries

This section investigates how to evaluate complex queries efficiently. Our reflections on the different kinds of queries are not independent of each other. Our proposals how to evaluate multi-feature and multi-object queries build on efficient evaluation techniques for simple queries. The approach for multi-feature multi-object queries is a generalization of the ones for multi-feature single-object queries and single-feature multi-object queries. Finally, since we expect IO-costs to dominate the costs of query evaluation, we will strive to keep these costs small. Reducing CPU costs will be an issue only when they are larger than IO costs. These subsections introduce the GeVAS algorithms step by step, together with the optimizations.

3.1 Single-Feature Single-Object Queries

Feature representations of images typically are high-dimensional. There is evidence [13, 1] that NN-search in high-dimensional spaces is effective and yields satisfactory

retrieval quality. However, the exact evaluation of such queries is linear in the number of data objects [16, 27, 4, 3]. A well accepted implementation of NN-search works with quantized representations of the points, as first described in [27] with the *vector-approximation file* (VA-File). Extensions of the method include the IQ-Tree [3] and the A-Tree [23]. In the following, we review the VA-File idea.

Structure of the VA-File. The VA-File consists of two files: one contains a short approximation of the feature representation of each data point, the other one the exact representations of each point. We obtain the quantizations by laying a grid over the data space and approximating the points by their surrounding cells (cf. Figure 2). Grid lines are chosen such that the number of objects between two neighboring lines is roughly the same. Note that we do not need a distance function to build the VA-File.

Subsequently, b denotes the number of approximation bits per dimension, and d is the dimensionality of the feature space. In Figure 2, $d = 2$, and $b = 2$. A realistic value for b would be between 6 and 8, as shown in [27].

Given the feature representation $\mathbf{q}$ of a query point, its distance to the cell of a point $\mathbf{p}$ is a lower bound of the distance of $\mathbf{p}$ and $\mathbf{q}$. Analogously, we can upper bound that distance (cf. right half of Figure 2). More formally, $m[j,0]$, ..., $m[j,2^b]$ are the partition points in dimension j, $s_j(\mathbf{p}) \in \{0,\ldots,2^b-1\}$ is the number of the interval of $\mathbf{p}$ in dimension j, i.e., the approximation of $\mathbf{p}$ for dimension j, and q_j is the value of $\mathbf{q}$ in dimension j. The lower bound of the distance of $\mathbf{p}$ and $\mathbf{q}$ is as follows, using the L_2 metric:

$$lBnd_2(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{j=0}^{d-1} \begin{cases} (m[j,s_j(\mathbf{p})]-q_j)^2 & q_j < m[j,s_j(\mathbf{p})] \\ (q_j - m[j,s_j(\mathbf{p})+1])^2 & q_j > m[j,s_j(\mathbf{p})+1] \\ 0 & otherwise \end{cases}} \tag{3.1}$$

VA-File based query evaluation. NN-query evaluation consists of two phases. The first phase (*filtering phase*) iterates through the approximation file and computes a lower and an upper bound on the distance of the current data point to the given query point (cf. Figure 2). These bounds allow the pruning of the vast majority of data objects. The second phase (*inspection phase*) then retrieves the exact representation of the remaining objects and computes their distance. More formally, the second phase inspects $\mathbf{p}$ if and only if $lBnd_2(\mathbf{p},\mathbf{q}) \leq nn^{dist}(\mathbf{q})$, where $nn^{dist}(\mathbf{q})$ denotes the NN-distance of $\mathbf{q}$. Experience shows that the number of data objects dealt with in the second phase is almost independent of the number of data objects N; a typical number is five objects for $k = 1$ and $N = 500,000$.

Implementation of VA-File based query evaluation. The following optimizations reduce computation costs:

1. The distances between the query point and each line of the grid are precomputed before the first phase. This simplifies computations of the bounds.
2. We avoid the s-th root when dealing with an L_s metric. Due to the monotonicity of the root-function, we still obtain the correct rank. We compute the correct distance only for the objects in the answer set.
3. We keep track of the k smallest upper bounds seen so far during the first phase. We can terminate the computation of the current lower bound when it is larger than those k upper bounds.

3.2 Multi-Feature Single-Object Queries

This subsection discusses evaluation of multi-feature single-object queries, i.e., there is one reference object with m features. Multi-feature single-object queries require normalization, as described in Subsection 2.2. We see two alternative evaluation schemes, *feature fusion* and *feature fusion on-the-fly*. Feature fusion works with both VA-File and trees as underlying data structure (even though we describe it only for the VA-File). The second scheme in turn is feasible with the VA-File only.

New but impractical approach: feature fusion. Indexes may apply to individual attributes or to combinations of attributes with relational databases. In analogy to the combination case, the idea with feature fusion is to concatenate the feature points representing a data object, always in the same order, and to build an index or a VA-File on these new points. As with simple queries, approximation-based data structures such as the VA-File are more appropriate than traditional hierarchical methods like the SR-Tree [16]. Namely, when concatenating several features, the overall dimensionality increases, and those traditional methods further deteriorate. Feature fusion avoids the random-access phase, in comparison to the A_0-algorithm [9]. However, this comes at additional storage costs.

While our query model allows for explicit references to different feature types plus weights, one might wonder why one 'large' feature that is a concatenation of all the individual features is not sufficient. At first sight, this seemingly leads to higher result quality. However, the additional flexibility of our model is necessary: The number of feature types available is large and the costs of evaluating a query grow with the sum of the number of dimensions. On the other hand, it depends on the concrete information need whether an additional feature type leads to better query results. A feature type may not be of much help if another type can more or less substitute it, if the similarity notion of the feature-extraction algorithm is too restrictive from a user perspective, or if the weight of a feature type chosen by a relevance-feedback model is very small. The respective feature may be left aside in such cases.

New approach: feature fusion on-the-fly. In contrast to feature fusion, the points corresponding to one data object are now brought together only in main memory during query evaluation. For each feature type, we build a VA-File such that the order of objects is the same in all files. Evaluation of multi-feature queries again consists of two phases. The first one scans through all approximation files in parallel. For each object, the approximations determine bounds on the normalized feature distances which are combined to bounds on the query distance. After pruning, the second phase accesses the exact representations of each remaining data object in the different vector files, computes its ex-

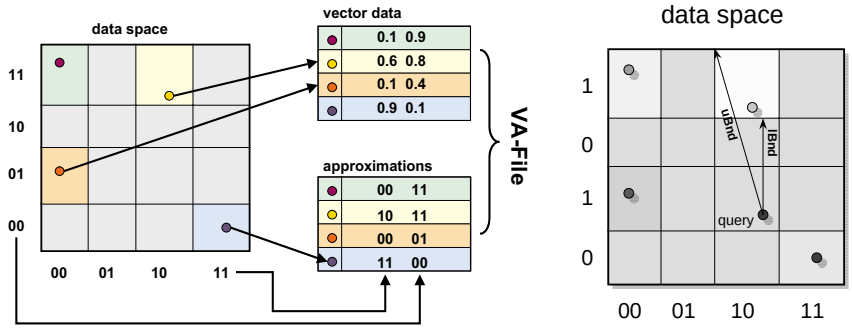


Figure 2: Structure of the VA-File

act query distance and returns the objects with the smallest query distance. Hence, search efficiency is almost equal to the one of feature fusion except for the second phase: instead of reading only k' points as with feature fusion, we have to read $k' \cdot m$ points, one for each feature type. These additional costs however are not significant, as our experiments will show. Consequently, feature fusion on-the-fly is almost as efficient as feature fusion, but without exorbitant additional storage costs.

It is important to notice that feature fusion on-the-fly is not feasible with hierarchical methods, i.e., both "classical" tree-based approaches [2, 16] and approaches extending the VA-File [3, 23]. This is because the leaf nodes of trees for different features do not contain the same objects.

3.3 Single-Feature Multi-Object Queries

We now investigate the evaluation of queries consisting of n reference objects with the same feature type. This subsection also pinpoints relationships between the query semantics and the techniques for query evaluation. $\delta(\mathbf{p},\Gamma)$ denotes the query distance in the single-feature multi-object query case where Γ is the set of feature representations of the query objects.

Existing approach: CPZ-approach. Tree-based implementations of simple NN-search compute a lower and upper bound on the distance between data points in a bounding region and the query point. They use these bounds to prune bounding regions and to decide on their order of inspection [14]. Furthermore, recall that a distance-combining function takes object distances as arguments and returns the query distance. The approach used in [8] is the computation of the bounds on the query distance of the bounding regions. They do so by applying the distance-combining function to the bounds on the object distances of the bounding regions. The bounds on the query distance are used for pruning and ordering the bounding regions, as in the simple case. In other words, NN-search traverses the tree with all query points in parallel.

New approach: VA-File for multi-object queries. It is straightforward to adapt the CPZ-approach to the VA-File: one determines the bounds on the distance between the current data point and each reference object. These bounds are then combined to the bounds on the query distance. These are indeed bounds as the distance-combining function is monotonic. The computational complexity of the first phase of the approach is $o(d \cdot N \cdot n)$. Obviously, if n is large enough, the computational costs may exceed the ones for reading the quantizations from disk. In some cases, however, it is possible to reduce the above complexity to $o(d \cdot (N+n))$ depending on the the metric in use and the distance-combining function. We will refer to this optimization as *precomputation of bounds*. It is a generalization of Optimization 1 in Subsection 3.1.

For instance, assume that we use the L_1-metric and Weighted Average to combine distances. Let $lBnd_j(\mathbf{q_i},\mathbf{p})$ denote the lower bound on the distance in dimension j between the feature point $\mathbf{q}_i$ of the i-th reference object and the one for the current data point $(\mathbf{p})$. It is given as:

$$lBnd_j(\mathbf{q_i},\mathbf{p}) = \begin{cases} m[j,s_j(\mathbf{p})] - (\mathbf{q_i})_j & (\mathbf{q_i})_j < m[j,s_j(\mathbf{p})] \\ (\mathbf{q_i})_j - m[j,s_j(\mathbf{p})+1] & (\mathbf{q_i})_j > m[j,s_j(\mathbf{p})+1] \\ 0 & otherwise \end{cases} \tag{3.2}$$

Leaving aside the weights for the ease of presentation, the combined lower bound is then as follows:

$$lBnd(\mathbf{p}) = \sum_{i=1}^{n}\sum_{j=1}^{d} lBnd_j(\mathbf{q_i},\mathbf{p}) = \sum_{j=1}^{d}\left(\sum_{i=1}^{n} lBnd_j(\mathbf{q_i},\mathbf{p})\right) \tag{3.3}$$

The term in parentheses in the last equation only depends on the grid slice in which the data point in dimension j lies, given a query. That term has the same value for all points in this slice. So we precompute these values for all dimensions and all slices. The number of slices is rather small, e.g., there are only 256 slices per dimension with 8 bits. Let $lBnd_j(s)$ be the precomputed combined lower bound for the s-th slice in dimension j. The computational complexity to precompute all $lBnd_j(s)$ is $o(d \cdot n)$. The computation of combined lower bounds simplifies to:

$$lBnd(\mathbf{p}) = \sum_{j=1}^{d} lBnd_j(s_j(\mathbf{p})) \tag{3.4}$$

An analogue scheme can be deployed to determine upper bounds on the combined distances, abbreviated with $uBnd$. Summing up, the first phase of VA-File based NN-search consists of two steps: 1) precomputing $lBnd_j(s)$ and $uBnd_j(s)$ for all dimensions and all slices, and 2) applying

Equation (3.4) to each data object. The overall complexity is thus: $o(d \cdot (N+n))$.

Unfortunately, this scheme is not always applicable. For instance, if we use Fuzzy-And to combine distances, we cannot change the order of max and $\sum$ (cf. Equation (3.3)). Analogously, we cannot deploy the precomputation scheme with the L_2-metric because of the square root. In these cases and if n is large, the computational costs may exceed the ones for IO. The rationale behind the following optimization is to avoid this for most settings.

Refinement of our new approach: early abort of distance combination. The discussion so far has insinuated that the GeVAS algorithms compute bounds on all distances of the current data object before applying the distance-combining function, if precomputation of bounds is not feasible. However, this is not necessary in many cases. When combining distances with *Weighted Average*, we can abort as soon as the left inequality holds (*maxDist* is the smallest upper bound seen so far):

$$\sum_{j=1}^{i} w_j \cdot x_j > maxDist \quad \Rightarrow \quad \sum_{j=1}^{n} w_j \cdot x_j > maxDist \qquad i < n$$

The implication holds because $w_j > 0$, and $x_j \geq 0$. $x_j \geq 0$ because we do not normalize.

Let us now look at *Fuzzy-And* as the distance-combining function. We can abort the computation of the combined distance if the maximum distance encountered so far exceeds the upper bound *maxDist*. Namely,

$$\max_{j=1}^{i} x_j > maxDist \quad \Rightarrow \quad \max_{j=1}^{n} x_j > maxDist \qquad i < n$$

With *Fuzzy-Or*, we cannot do much. We have to compute *all* distances and cannot prune according to *maxDist*. But $\min(maxDist, x_1, \ldots, x_{i-1})$ is an upper bound for x_i.

Our implementation covers further cases, but explicitly dealing with them here does not provide any additional insight. Finally, early abort of distance combination is also feasible with multi-feature queries. But IO-costs dominate with such queries, and the effect of the optimization is limited. The following table lists all combinations of metric and distance-combining function and indicates whether a precomputational scheme is available (++), or, alternatively, whether early abort of distance combination is expected to yield significant cost reduction (+):

distance-combining func	L_1	L_2	$(L_2)^2$	L_∞
Weighted Average	++	+	++	+
Fuzzy-Or	0	0	0	0
Fuzzy-And	+	+	+	++

3.4 Multi-Feature Multi-Object Queries

Finally, we deal with the general case, again elaborating the relationship between similarity measure and performance. To evaluate multi-feature multi-object queries with GeVAS, the techniques from the previous two subsections need to be combined. While the combination is relatively straightforward in some cases, the obvious approach leads to IO-costs that are unnecessarily high in the general case.

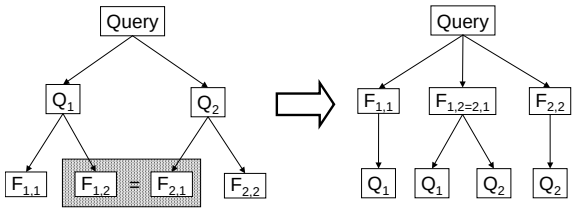


Figure 3: Restructuring a complex query.

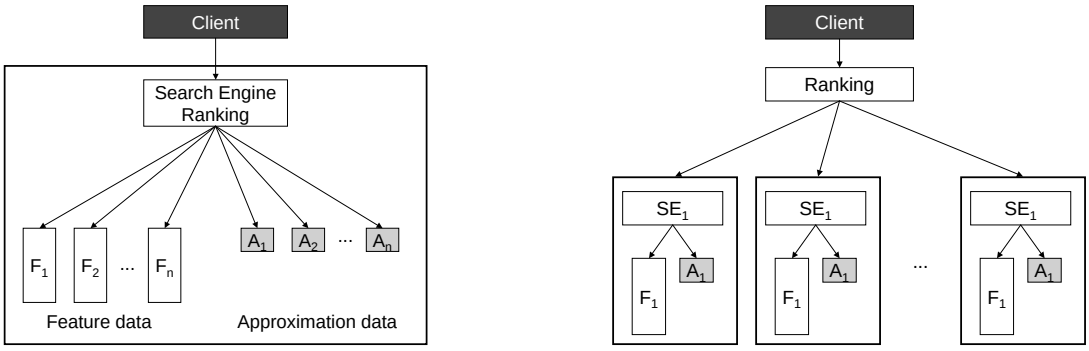
Optimization: Shared iterators. The sets of feature types of different reference objects may overlap. Rather than reading the approximation file of each feature type several times, the ideal evaluation scheme has the following characteristic: one iteration over the approximation file for a certain feature type evaluates all sub-queries that refer to that feature type. We call this technique *shared iterators*. It is feasible independent of the metric and the distance-combining function in use.

New approach: Restructuring of the query. For multi-object queries, we have developed an optimization that precomputes bounds. Unfortunately, it is not applicable with the query in Figure 3 (left). The reason is that the color feature distances of the data objects to Q_1 and Q_2 are not directly combined with each other. We can avoid this effect with an additional measure. If the complex query only uses one type of distance-combining functions, we can restructure the query from a set of multi-feature queries into a set of multi-object queries. As a result, the transformed query contains a multi-object query for each feature type. Our evaluation technique of the last subsection (*precomputation of bounds*) can handle it efficiently. Unfortunately, restructuring of a query is not feasible if it uses different types of distance-combining functions.

4 Quality of Bounds

Prior to conducting performance studies, it is not evident whether the performance of GeVAS will be as good as the one of VA-File based search for simple queries. [25]has investigated the relationship of the number of bits, the approximation error and the expected performance of the VA-File. Our investigation showed that the VA-File is faster than a sequential scan or a tree, if the number of candidates in the second phase is small, e.g. smaller than 100. Otherwise, the second phase invokes too many random accesses to the vector data, and performance deteriorates, much like with trees. Furthermore, we have found that the number of candidates is directly related to the error of the approximations, i.e. the difference between the bounds and the distance. For instance, increasing the number of bits by one, halves the error of the bounds. This reduces the number of candidates significantly.

These relationships also hold in the complex case. However, it is not obvious how many bits are required to keep the bounds tight, i.e. whether we need more bits than in the simple case. The number of candidates would become so large with a bad configuration that the second phase lasts



(a) GeVAS (b) A_0-algorithm

Figure 4: Different architectures for query evaluation.

longer than a sequential scan over the exact vector representations. In analogy to the simple case studied in [25], we have investigated the influence of the number of query objects on the error of bounds and on the number of candidates [5]. Our approach is as follows: we show that the error of bounds is at most twice as large as in the simple case with an infinite number of query objects using the same number of bits in the worst case. Thus, we can achieve the same tight bounds for a multi-object query as for a simple query by increasing the number of bits per dimension by at most 1. In the case of multi-feature queries or multi-feature multi-object queries, we may regard the different features as one large feature. Thus, the number of bits only increases logarithmically with the total number of dimensions [25]. For instance, if the number of dimensions increases by a factor of 4, we need one additional bit per dimension to keep the number of candidates constant.

5 Experimental Setup

The remainder of this article reports our experimental evaluation. The evaluation includes all query-evaluation techniques this article has mentioned. Our implementation of the A_0-algorithm [9] uses the VA-File [27] as the indexing method for the sources. In the case of multi-object queries, we extended the R*-Tree [2], the SR-Tree [16] and the M-Tree [7] according to the CPZ-approach. These tree-based evaluation alternatives are 'fully optimized'.

Figure 4 illustrates the architectures of our approach and the one of the A_0-algorithm. The A_0-algorithm obviously is on top of search components, while our approach integrates all feature data into one component. One may argue that the architecture in Figure 4 (b) lends itself to parallelization in a cluster of workstations while the one in Figure 4 (a) is limited to a single machine. However, we already have reported on a parallel implementation of the VA-File [26]. Our results have shown that the speedup is roughly linear in the number of components. An analogous extension for the complex case is straightforward (and already implemented) and achieves the same speedup. In what follows, we conducted all experiments on a single machine.

Hardware. All experiments ran on a PentiumIII with 450 MHz and 512 MBytes main memory. The data was stored on a SCSI Harddisk with an average seek time of 7 ms, and a data transfer rate of 16 MB/s. We cached all the internal nodes with the tree-based index methods. Only the leaves were read directly from disk, bypassing the file cache of the system. All data was fetched directly from the disk without any caching with the VA-File.

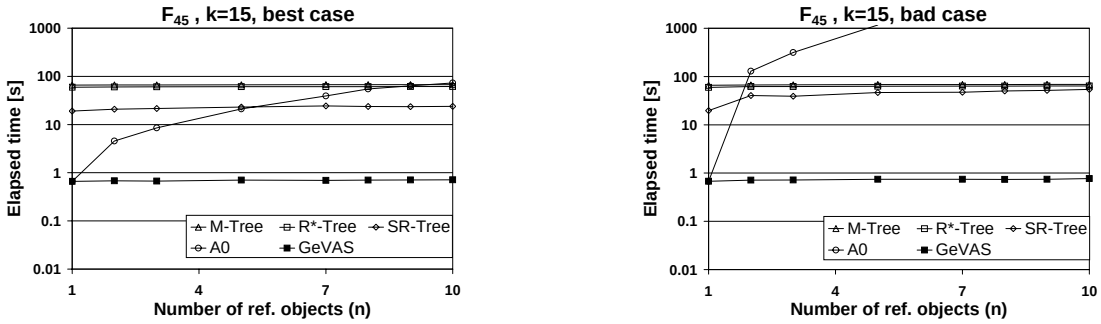
Data Sets and Further Settings. We have gathered around 230,000 images from diverse sources. We have extracted the following feature types for each of these images: 1) color moments in the Lab-color space [24], subsequently abbreviated with F_9 or F_{45}, 2) RGB color histograms with 64 bins (F_{64}) [20], and 3) texture moments based on gabor filters (F_{30}) [17]. The number in the abbreviations stands for the dimensionality of the respective features. The following experiments only use these real data sets and no synthetic ones. The page size of the tree algorithms was 8 kB throughout all experiments. We always used 8 bits per dimension for the VA-File. All experiments measured average times for the 15-NN-search.

We distinguish between two scenarios with multi-object queries: a best case scenario and a bad case scenario. The first scenario uses the objects in the answer set of a previous NN-query as the reference objects. This corresponds to the situation where relevance feedback constructs the query. The bad case scenario randomly selects n points from the data set. With both scenarios, we used the Weighted Average distance-combining function and the $(L_2)^2$ distance measure. Experiments with other functions and measures yielded almost identical results.

6 Experiments

6.1 Single-Feature Multi-Object Queries

Figure 6.1 depicts the elapsed times of the approaches as a function of n for the best case scenario for a real data set with 45 dimensions and 230,000 objects, using a logarith-



(a) (b)

Figure 5: Elapsed times for multi-object queries. (a) Best case; and (b) bad case.

mic time scale. The A_0-algorithm is not bad for small n and even beats some of the trees (due to the superiority of the VA-File). However, search costs "explode" with the number of reference objects. Remember, however, that the approaches are not fully compatible. For all other methods, elapsed times are mostly independent of n, as IO is the dominant cost factor. GeVAS outperforms the extended trees by one to two orders of magnitude.

In the bad case scenario depicted in Figure 6.1, A_0 performs poorly: search costs with two reference objects are more than 100 times larger than with only one. The main reason is that the A_0-algorithm must fetch a large number of items from each source until 15 matches are found. For instance, in the bad case for $n = 5$, the A_0-algorithm randomly accessed 10 percent of all data objects, leading to a response time of around twenty minutes in our setup. This dramatic deterioration was already mentioned and explained in [9]. The difference between QuickCombine and A_0, as reported in [11], is much smaller than the one between GeVAS and the A_0-algorithm. Performance of the M-Tree and the R*-Tree is about equal for varying numbers of reference objects as they access all leaf pages, but poor in absolute terms (two orders of magnitude below the one of GeVAS). Only the SR-Tree performs worse in the bad case than in the best case as it was the only tree capable to prune some leaf pages in the best case. GeVAS executes the queries in the bad case as fast as in the best case.

The next series of experiments leaves the number of reference objects constant ($n = 5$), and varies the data sets. We measured the elapsed times and determined the improvement factor of GeVAS over the other approaches. Figure 6 shows the respective results. Note that the factor scale is logarithmic, and bars with values above 1 mean that GeVAS performs better by the factor shown. Figures 6.1 and 6.1 contain the graphs for the elapsed times of the best case scenario and the bad case scenario, respectively. In the bad case, GeVAS typically is faster by a factor of at least 10 than the other approaches. When comparing to A_0, the factor easily exceeds 1000. In the best case, only the SR-Tree with the data set F_9 is competitive to GeVAS, as the dimensionality of this data set is only 9.

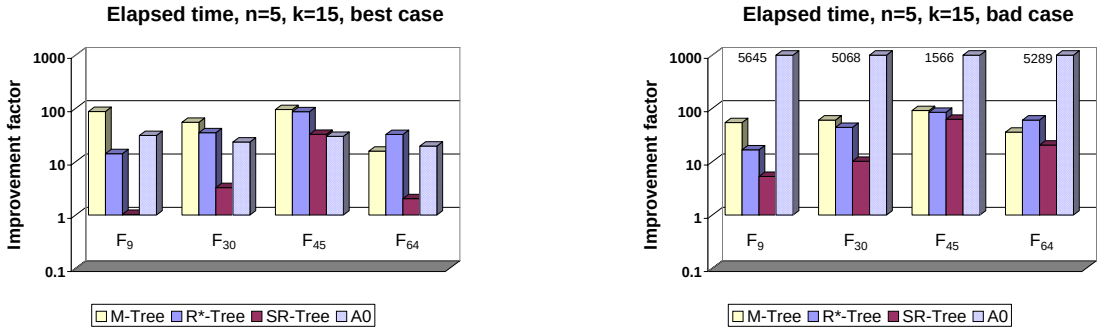
For quantification of the speedup due to the precomputation of bounds described in Section 3.3, we compared the elapsed times and the CPU times of an optimized GeVAS implementation with a non-optimized one. We used the $(L_2)^2$ metric, Weighted Average as distance combining function and the F_{45} feature. We added the times of the SR-Tree with early abort of distance computation as a reference point. Figure 7 depicts the elapsed times and CPU times as a function of the number of reference objects. Even the unoptimized VA-File-based algorithm performs always better than the one based on the SR-Tree by at least a factor of 3 with regard to elapsed time. Comparing the elapsed times (left diagram) with the CPU times (right diagram) of the non-optimized algorithm, computational costs dominate the response time for $n > 5$. With the optimized algorithm, computational costs are almost constant, i.e., only the costs for precomputing the bounds grow with n. This means that search costs are bound by the IO-costs. In other words, the number of reference objects does not affect overall response time.

6.2 Multi-Feature Single-Object Queries

This subsection compares the performance of a) feature fusion, b) feature fusion on-the-fly, and c) Fagin's A_0-algorithm for multi-feature queries. All three cases deploy the VA-File. We will abbreviate (b) with VAF-single and (c) with Single-A0. In contrast to the multi-object case, multi-feature queries are closer to best-case queries than to bad-case queries. This is because many feature types are correlated, and features rank the data objects more or less in comparable order. Figure 6.3 graphs elapsed times for the various approaches for different combinations of two feature types. The A_0-algorithm performs well, but worse than feature fusion. Feature fusion on-the-fly is slightly worse than feature fusion. We conclude that search costs depend on the total number of dimensions. Only a small portion of the costs is due to the access to several files.

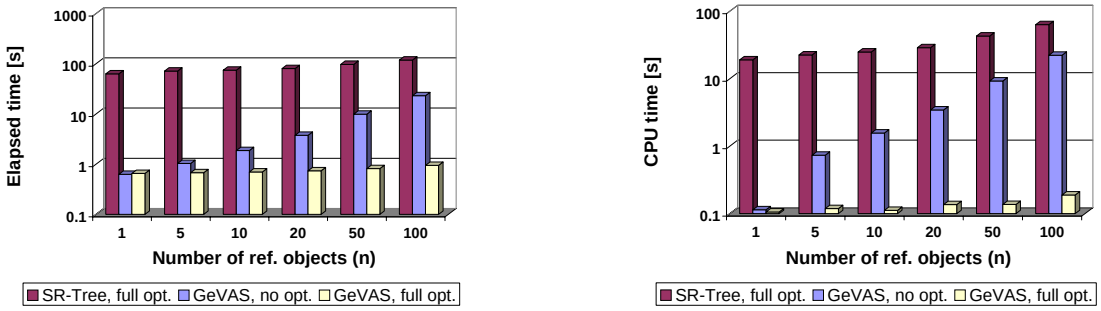
6.3 Multi-Object Multi-Feature Queries

Finally, we quantify the benefit of query restructuring. Figure 6.3 graphs elapsed time using two feature types, i.e., each reference object refers to the same two feature



(a) (b)

Figure 6: Multi-object queries – improvement factor of elapsed time. (a) Best case; and (b) bad case.



(a) (b)

Figure 7: Effect of the optimizations with multi-object queries. (a) elapsed times; (b) CPU times.

types. Furthermore, we again distinguish between a best case and a bad case scenario and between an evaluation with restructuring ("full opt.") and a non-optimized evaluation ("no opt."). Obviously, restructuring keeps the response times almost constant. Even with 100 reference objects, elapsed times are not much larger than with only one reference object. Moreover, there is no difference between a bad case and a best case, i.e., the optimized evaluation of complex queries is very stable.

7 Conclusions

Complex similarity queries yield better results than simple ones. Formulating such complex queries in a natural way is easy when using relevance-feedback mechanisms. The topic of this article has been the efficient evaluation of such queries. We have combined/generalized existing approaches, resulting in a set of techniques referred to as Generalized VA-File-based Search (GeVAS). We see our main contributions in showing that GeVAS is feasible, and in investigating the relationship between query semantics and efficiency of query evaluation. This has given rise to various optimizations that are not evident, or whose benefits are not evident. We have evaluated our techniques and the refinements using a full-fledged prototype over large data sets. We have shown that the optimizations give rise to a significant improvement in most cases. Future work will investigate the transformation of relevance judgements into the various forms of multi-object multi-feature queries.

Online Demo and Extended Version. We have implemented the GeVAS set of algorithms as described in this paper, and used them to enable relevance feedback in our large image retrieval system [15]. The current system contains more than 220,000 images, deploys around 20 different feature types with dimensionalities between 9 and 560, and features relevance feedback with various models. This requires the full complexity of the query model described in this paper. Finally, there is an extended version of this paper covering additional aspects like the quality of bounds for complex queries [5].

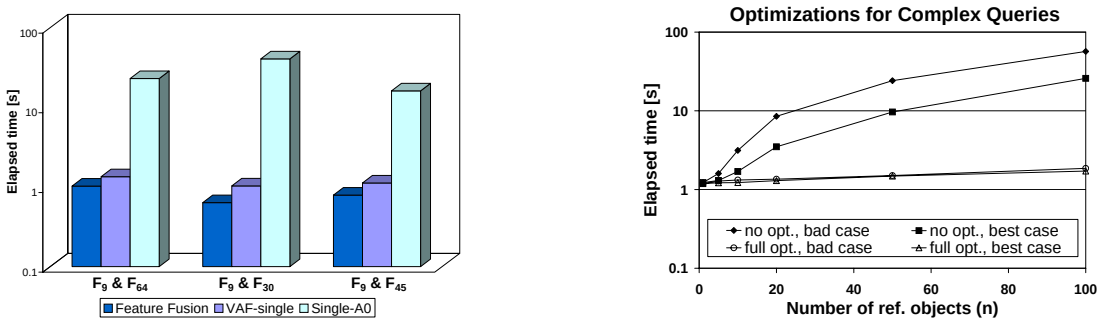
Acknowledgments. We thank the member of the database group at ETH, notably Torsten Grabs, Uwe Röhm, Heiko Schuldt, and Can Türker for good comments on earlier versions of this article.

References

[1] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proc. of the Int. Conf. on Database Theory*, volume 1973 of *Lecture Notes in Computer Science*, p. 420–434, London, UK, 2001.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, p. 322–331, Atlantic City, NJ, USA, 1990.

[3] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent Quantization: An Index Compression Technique for



(a) (b)

Figure 8: Elapsed times for (a) 2-feature queries and (b) multi-feature multi-object queries.

High-Dimensional Data Spaces. In *Pro. of the Int. Conf. on Data Engineering*, p. 577–588, San Diego, CA, 2000.

[4] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "Nearest Neighbour" Meaningful? In *Proc. of the Int. Conf. on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, p. 217–235, Jerusalem, Israel, 1999.

[5] K. Böhm, M. Mlivoncic, H.-J. Schek, and R. Weber. Fast Evaluation Techniques for Complex Similarity Queries. Technical report, Dept. of Computer Science, 2001. Available at http://www-dbs.ethz.ch/∼weber/paper/VLDB01Long.ps.

[6] K. Chakrabarti, K. Porkaew, and S. Mehrotra. Efficient Query Refinement in Multimedia Databases. In *Pro. of the Int. Conf. on Data Engineering*, p. 196, San Diego, California, USA, 2000.

[7] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the Int. Conference on Very Large Databases*, p. 426–435, Athens, Greece, 1997.

[8] P. Ciaccia, M. Patella, and P. Zezula. Processing Complex Similarity Queries with Distance-Based Access Methods. In *Proc. of the Int. Conf. on Extending Database Technology*, volume 1377 of *Lecture Notes in Computer Science*, p. 9–23, Valencia, Spain, 1998.

[9] R. Fagin. Combining Fuzzy Information from Multiple Systems. In *Proc. of the ACM Symposium on Principles of Database Systems*, p. 216–226, Montreal, Canada, 1996.

[10] R. Fagin and E. L. Wimmers. A Formula for Incorporating Weights into Scoring Rules. In *Proc. of the Int. Conf. on Database Theory*, volume 1186 of *Lecture Notes in Computer Science*, p. 247–261, Delphi, Greece, 1997.

[11] U. Güntzer, W.-T. Balke, and W. Kiessling. Optimizing Multi-Feature Queries for Image Databases. In *Proc. of the Int. Conference on Very Large Databases*, p. 419–428, Cairo, Egypt, 2000.

[12] D. Harman. *Information Retrieval: Data Structures and Algorithms*, chapter 11: Relevance Feedback and Other Query Modification Techniques, p. 241–263. Prentice Hall, Englewood Cliffs, New Jersey, 1992.

[13] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What Is the Nearest Neighbor in High Dimensional Spaces? In *Proc. of the Int. Conference on Very Large Databases*, p. 506–515, Cairo, Egypt, 2000.

[14] G. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proceedings of the Fourth International Symposium on Advances in Spatial Database Systems (SSD95)*, number 951 in Lecture Notes in Computer Science, p. 83–95, Portland, Maine, 1995.

[15] http://simulant.ethz.ch/Chariot/, 2000.

[16] N. Katayama and S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, p. 369–380, Tucson, Arizon USA, 1997.

[17] B. Manjunath and W. Ma. Texture Features for Browsing and Retrieval of Image Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–842, 1996.

[18] H. Müller, W. Müller, D. M. Squire, and S. Marchand-Maillet. Strategies for positive and negative relevance feedback in image retrieval. In *Proceedings of the International Conference on Pattern Recognition (ICPR'2000)*, volume 1 of *Computer Vision and Image Analysis*, p. 1043–1046, Barcelona, Spain, 2000.

[19] S. Nepal and M. Ramakrishna. Multi Feature Query By Multi Examples in Image Databases. In *Proceedings of the Tenth International Conference on Management of Data*, Pune, India, 2000.

[20] W. Niblack, R. Barber, W. Equitz, et al. The QBIC Project: Querying Images by Content, Using Color, Texture, and Shape. In *Storage and Retrieval for Image and Video Databases*, volume 1908 of *SPIE Proceedings*, p. 173–187, San Jose, CA, USA, 1993.

[21] J. Rocchio Jr. *Relevance Feedback in Information Retrieval, The SMART Retrieval System: Experiments in Automatic Document Processing*, chapter 14, p. 313–323. Prentice Hall, Englewood Cliffs, New Jersey, USA, 1971.

[22] Y. Rui, T. Huang, and S. Mehrotra. Relevance Feedback Techniques in Interactive Content-Based Image Retrieval. In *Storage and Retrieval for Image and Video Databases*, p. 25–36, San Jose, California, USA, 1998.

[23] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima. The A-tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. In *Proc. of the Int. Conference on Very Large Databases*, p. 516–526, Cairo, Egypt, 2000.

[24] M. Stricker and A. Dimai. Spectral Covariance and Fuzzy Regions for Image Indexing. *Machine Vision and Applications*, 10:66–73, 1997.

[25] R. Weber and K. Böhm. Trading Quality for Time with Nearest-Neighbor Search. In *Proc. of the Int. Conf. on Extending Database Technology*, volume 1777 of *Lecture Notes in Computer Science*, p. 21–35, Konstanz, Germany, 2000.

[26] R. Weber, K. Böhm, and H.-J. Schek. Interactive- Time Similarity Search for Large Image Collections Using Parallel VA-Files. In *Research and Advanced Technology for Digital Libraries, 4th European Conference (ECDL)*, volume 1923 of *Lecture Notes in Computer Science*, p. 83–92, Lisbon, Portugal, 2000.

[27] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. of the Int. Conference on Very Large Databases*, p. 194–205, New York City, New York, USA, 1998.

[28] E. L. Wimmers, L. M. Haas, M. T. Roth, and C. Braendli. Using Fagin's Algorithm for Merging Ranked Results in Multimedia. In *Proceedings of the Fourth IFCIS International Conference on Cooperative Information Systems*, p. 267–278, Edinburgh, Scotland, 1999.