FeedbackBypass: A New Approach to Interactive Similarity Query Processing

Ilaria Bartolini Paolo Ciaccia Florian Waas

DEIS – CSITE-CNR, University of Bologna
Bologna, Italy

Abstract

In recent years, several methods have been proposed for implementing interactive similarity queries on multimedia databases. Common to all these methods is the idea to exploit user feedback in order to progressively adjust the query parameters and to eventually converge to an "optimal" parameter setting. However, all these methods also share the feature to "forget" user preferences across multiple query sessions, thus requiring the feedback loop to be restarted for every new query (i.e. using default parameter values), which not only is frustrating from the user's point of view but is also a significant waste of system resources.

In this paper we present FeedbackBypass, a new approach to interactive similarity query processing. It complements the role of relevance feedback engines by storing and maintaining the query parameters, determined with feedback loops over time, using a wavelet-based data structure called "Simplex Tree". For each query, a favorable set of query parameters can be determined using the Simplex Tree. This information can then be used to either "bypass" the feedback loop completely for already-seen queries, or to start the search process from a near-optimal configuration.

FeedbackBypass can be well combined with all state-of-the-art relevance feedback techniques working in high-dimensional vector spaces. Its storage requirements scale linearly with the dimensionality of the query space, making even sophisticated query spaces amenable. Experimental results demonstrate both the effectivness and efficiency of our technique.

1 Introduction

Similarity and distance-based queries are a powerful way to retrieve interesting information from large multimedia repositories [Fal96]. However, the very nature of multimedia objects often complicates the user's task of choosing an appropriate query and a suitable distance criterion to retrieve from the database the objects which best match his/her needs [SK97]. This can be due both to limitation of the query interface and to the objective difficulty, from the user's point of view, to properly understand how the retrieval process works in high-dimensional spaces, which typically are used to represent the relevant *features* of the multimedia objects . For instance, the user of an image retrieval system will hardly be able to predict the effects that the modification of a single parameter of the distance function used to compare the individual objects can have on the result of a query.

To obviate this unpleasant situation, several multimedia systems now incorporate some *feedback* mechanisms so as to allow users to provide an evaluation of the *relevance* of the result objects. By properly analyzing such relevance judgments, the system can then generate a new, refined query, which will likely improve the quality of the result, as experimental evidence confirms [RHOM98]. This interactive

retrieval process, which can be iterated several times until the user is satisfied with the results, gives rise to a so-called *feedback loop* during which the default parameters used by the query engine are gradually adjusted to fit the user's needs (see e.g. [ORC+97]).

Although relevance feedback has been recognized as a highly effective tool, its applicability suffers two major problems:

1. Depending on the query, it may require numerous iterations before an acceptable result is found, thus convergence can be slow.

2. Once the feedback loop of a query is terminated, no information about this particular query is retained for re-use in further processing. Rather, for further queries, the feedback process is started anew with default values. Even in the case that a query object has already been used in an earlier feedback loop, all iterations have to be repeated.

Note that both problems concern the *efficiency* of the feedback process, whereas its *effectivenes* will depend on the specific feedback mechanisms used by the system, on the similarity model, and on the features used to represent the objects.

This paper presents FeedbackBypass, a new approach to interactive similarity query processing, which complements the role of current relevance feedback engines. FeedbackBypass is based on the idea that, by properly storing and maintaining the information on query parameters gathered from past feedback loops, it is possible to either "bypass" the feedback loop completely for already-seen queries, or to "predict" near-optimal parameters for new queries. In both cases, as an overall effect, the number of feedback and database search iterations is greatly reduced, resulting in a significant speed-up of the interactive search process.

Figure 1 shows a query image together with the 5 best results obtained from searching with default parameters a data set of about 10,000 color images. No result image belongs to the same semantic category of the query image, which is "Mammal" (see Section 6 for a description of image categories). The bottom line of the figure shows the 5 best matches obtained for the same query when FeedbackBypass has been switched-on, and the system uses the predicted query parameters. This leads to have 4 relevant images (i.e. 4 mammals) in the 5 top positions of the result.

The implementation of FeedbackBypass is based on a novel wavelet-based data structure, called *simplex tree*, whose storage overhead is linear in the dimensionality of the query space making even sophisticated query spaces amenable. Its resource requirements are *independent* of the number of processed queries but rely only on the query parameter function, which guarantees proper scalability and

Figure 1: FeedbackBypass in action. The top line shows the 5 best matches computed using default parameters for the query image on the left. The bottom line shows the results obtained for the same query when the parameters suggested by FeedbackBypass are used

performance levels. Furthermore, storage requirements can be easily traded-off for the accuracy of the prediction. Experimental results demonstrate both the effectiveness and efficiency of our technique.

The rest of the paper is organized as follows. Section 2 provides the background on relevance feedback mechanisms and on related work. In Section 3 we describe our approach and state the basic requirements for an effective implementation of FeedbackBypass. Section 4 shortly describes the underlying principles on which the Simplex Tree is based, and Section 5 provides a thorough description of the Simplex Tree and of related implementation issues. Experimental results on a real-world image data set are presented in Section 6. Section 7 concludes the paper.

2 Background and Related Work

We frame our discussion within the context of *vector space* similarity models, for which a multimedia object is represented through a D-dimensional vector (i.e. a point in $\Re^D$) of *features*, $\mathbf{p} = (p_1, \ldots, p_D)$, and the similarity of two points $\mathbf{p}$ and $\mathbf{q}$ is measured by means of some *distance function* on such space. Relevant examples of distance functions include L_p norms:

$$L_p(\mathbf{p}, \mathbf{q}) = \left(\sum_{i=1}^{D} (p_i - q_i)^p \right)^{1/p} \qquad 1 \leq p < \infty$$
$$L_\infty(\mathbf{p}, \mathbf{q}) = \max_i \{|p_i - q_i|\}$$

(L_1 is the Manhattan distance, L_2 is the Euclidean norm, L_∞ is the "max-metric") and their weighted versions. For instance, the weighted Euclidean distance is:

$$L_{2W}(\mathbf{p}, \mathbf{q}; \mathbf{W}) = \left(\sum_{i=1}^{D} w_i \left(p_i - q_i \right)^2 \right)^{1/2} \tag{1}$$

Further, *quadratic* distances can also be used to capture correlations between different coordinates of the feature vectors. The well-known Mahalanobis distance is defined as

$$d^2_{Mahalanobis}(\mathbf{p}, \mathbf{q}; \mathbf{W}) = \sum_{i=1}^{D} \sum_{j=1}^{D} w_{i,j} (p_i - q_i)(p_j - q_j)$$

and leads to arbitrarily-oriented ellipsoidal iso-distant surfaces in feature space [SK97]. Note that this distance is indeed a "rotated" weighted Euclidean norm.

The typical interaction with a multimedia retrieval systems that implements relevance feedback mechanisms can be summarized as follows [Sal88]:

Query formulation. The user submits an initial query $Q = (\mathbf{q}, k)$, where $\mathbf{q}$ is called the *query point* and k is a limit on the number of results to be returned by the system.

Query processing. The query point $\mathbf{q}$ is compared with the database objects by using a (default) distance function d. Then, the k objects which are closest to $\mathbf{q}$ according to d, $Result(Q, d) = \{\mathbf{p_1}, \ldots, \mathbf{p_k}\}$, are returned to the user.

Feedback loop. The user evaluates the relevance of the objects in $Result(Q, d)$ by assigning to each of them a *relevance score*, $Score(\mathbf{p_i})$. On the basis of such scores a new query, $Q' = (\mathbf{q}', k)$, and a new distance function, d', are computed and then used to determine the second round of results.

Termination. After a certain number of iterations, the loop ends, the final result being $Result(Q_{opt}, d_{opt})$, where $Q_{opt} = (\mathbf{q_{opt}}, k)$ is the "optimal" query the user had in mind, and d_{opt} the "optimal" way to retrieve relevant objects for Q_{opt}.

Each interactive retrieval system provides a specific implementation for each of the above steps. For instance, the choice of the initial query point depends on the system interface and, also considering the very nature of the multimedia objects, can include a *query-by-sketch* facility, the choice from a random sample of objects, the upload of the query point from a user's file, etc. Many options are also available for implementing the query processing step, which typically exploits index structures for high-dimensional data, such as the X-tree [BKK96] and the M-tree [CPZ97].

(a) (b)

Figure 2: The “query point movement” (a) and the ”re-weighting” (b) feedback strategies

More relevant to the present discussion are the issues concerning the feedback loop. The use of *binary* relevance scores is the simplest one, even from the user’s point of view. In this case the user can mark a result object either as “good” or “bad”, and implicitly assigns a neutral (“no-opinion”) score to non-marked objects. Graded, and even continuous, score levels have also been used to allow for a finer tuning of user’s preferences [RHOM98].

The two basic strategies for implementing the feedback loop concern the computation of a new query point (*query point movement*) and the change of the distance function, which can be accomplished by modifying the weights (importance) of the feature components (*re-weighting*).

Query point movement. The idea of this strategy is to try to move the query point towards the “good” matches (as evaluated by the user), as well as to move it far away from the “bad” result points (see Figure 2 (a)). A well-known implementation of this idea dates back to Rocchio’s formula [Sal88], which has been successfully used for document retrieval. More recently, query point movement has been applied by several image retrieval systems, such as the MARS system [RHOM98]. Ishikawa et al. [ISF98] have proved that, when using *positive* feedback (scores) and the Mahalanobis distance, the “optimal” query point (based on the available set of results) is a weighted average of the good results, i.e.:

$$\mathbf{q}' = \frac{\sum_i Score(\mathbf{p_i}) \times \mathbf{p_i}}{\sum_i Score(\mathbf{p_i})} \tag{2}$$

Re-weighting. The idea of re-weighting stems from the observation that user feedback can highlight that some feature components are more important than others in determining whether a result point is “good” or not, thus such components should be given a higher relevance. For simplicity of exposition, let us consider a retrieval model based on weighted Euclidean (see Equation 1) and also refer to Figure 2 (b). In order to assess the relative importance of the i-th feature vector component, the distribution of the “good” $p_{j,i}$ values, i.e. the values of the good matches along the i-th coordinate, is analyzed. In an earlier version of the MARS system [RHOM98], it was proposed to assign to the i-th coordinate a

weight w_i computed as the inverse of the standard deviation of the $p_{j,i}$ values, i.e. $w_i = 1/\sigma_i$. Later on, it was proved in [ISF98] that the "optimal" choice of weights is to have

$$w_i \propto \frac{1}{\sigma_i^2} \tag{3}$$

Similar results have been proved for quadratic distance functions [ISF98], as well as for the case where the number of good matches is less than the dimensionality of the feature space [RH00].

In a recent paper [RH00] Rui and Huang have extended the re-weighting strategy to a "hierarchical model" of similarity, where above strategy is first individually applied to each feature separately, and then each feature (rather than each feature component) is assigned a weight which takes into account the overall distance that good matches have from the query point by considering only that feature. Note that for F features this amounts to define the distance bewteen objects $\mathbf{p}$ and $\mathbf{q}$ as a weighted sum of feature distances, each of which the authors assume to have a quadratic form [RH00].

3 The FeedbackBypass Approach

This section explains in detail how FeedbackBypass can be smoothly integrated with query and relevance feedback models commonly used for similarity search.

The basic idea of our approach is to "bypass" the loop iterations of a typical interactive similarity retrieval system by trying to "guess" what the user is actually looking for, based only on the initial query he/she submits to the system.

If we abstract from the specific differences existing between the systems and concentrate on what all such systems share, two important observations can be done:

1. All systems assume that the user has in mind an "optimal" query point as well as an "optimal" distance function for that query.

2. Each time a new distance function is computed, this is taken from a *parameterized class* of functions (e.g. the class of weighted Euclidean distances), by appropriately setting the values of the class parameters.

This general state of things can be synthetically represented as a mapping:

$$\mathbf{q} \mapsto (\mathbf{q_{opt}}, d_{opt}) \equiv (\mathbf{q_{opt}}, \mathbf{W_{opt}}) \tag{4}$$

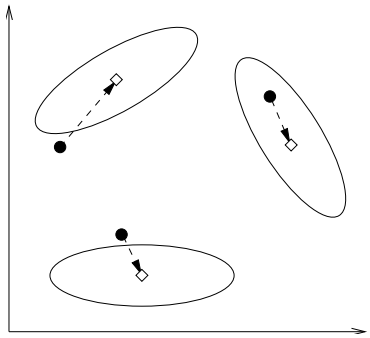


Figure 3: The optimal query mapping for 3 sample query points, assuming the class of Mahalanobis distances

which assigns to the initial query point $\mathbf{q}$ an optimal query point, $\mathbf{q_{opt}}$, and an optimal distance function, d_{opt}. The equivalence just highlights that d_{opt} is the distance function obtained when the parameters are set to $\mathbf{W_{opt}}$.

Figure 3 provides an intuitive graphical representation of the above mapping for three 2-dimensional query points. FeedbackBypass is based on the observation that, as more and more query points are added, an "optimal" *query mapping*, M_{opt}, from query points to query points and distance functions, will take shape, and that "learning" such mapping can indeed lead to "bypass" the feedback loop.

Let $\mathcal{Q} \subseteq \Re^D$ be the set of all query points and let $\mathcal{W}$ be the set of possible parameter choices, where each $\mathbf{W} \in \mathcal{W}$ corresponds to a distance function in the considered class. Then, the problem faced by FeedbackBypass can be precisely formulated as follows:

Problem 1 *Given the set $\mathcal{Q}$ of all possible query points and a class of distance functions with set of parameters $\mathcal{W}$, "learn" the query mapping $M_{opt} : \mathcal{Q} \rightarrow \mathcal{Q} \times \mathcal{W}$ which associates to each query point $\mathbf{q} \in \mathcal{Q}$ the "optimal" pair $(\mathbf{q_{opt}}, \mathbf{W_{opt}}) = M_{opt}(\mathbf{q})$.*

In other terms, the problem to be faced is that of learning the "optimal" way to map (query) points of $\Re^D$ into points of $\Re^{D+P}$, where P is the number of *independent* parameters that characterize a distance function in the chosen class. In the case when (query) points are normalized, the dimensionality of both the input (feature) and the output space of M_{opt} is reduced by 1.

Of course, statistical techniques for *dimensionality reduction*, such as the Karhunen-Loeve (K-L) transform, could be applied to lower the dimensionality of both the input and the output space. We do not consider dimensionality reduction in this paper, and leave it as an interesting follow-up of our research.

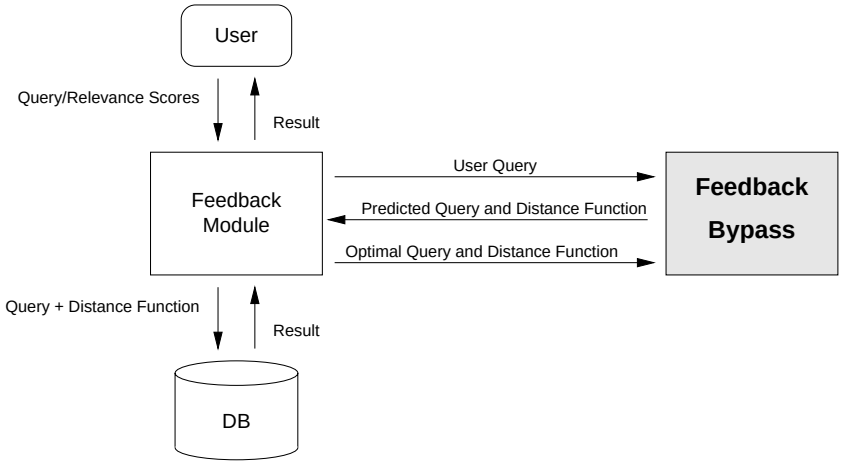


Figure 4: An interactive retrieval system enriched with the FeedbackBypass module

Example 1 Assume that objects are color images, which are represented by using a 32-bins color histogram, and that similarity is measured by the weighted Euclidean distance. Since the sum of the color bins is constant (it equals the number of pixels in the image) and one of the weights of the distance function can be set to a constant value, say 1, without altering at all the retrieval process, it turns out that M_{opt} is a function from $\Re^{31}$ to $\Re^{31+31}$. □

Figure 4 shows the basic architecture of an interactive retrieval system enriched with FeedbackBypass. Upon receiving the initial user query, the system forwards the query to FeedbackBypass, which returns a predicted optimal query point and the parameters for setting up the distance function. Then, the usual query processing-user evaluation-feedback computation loop can take place. When the loop ends, possibly after no iterations at all, the new feedback-determined query parameters for that query are passed to FeedbackBypass and stored for later reuse. Clearly, the storage phase can be skipped if no new feedback information has been provided by the user.

3.1 Requirements

The method we seek for learning M_{opt} from sample queries has to satisfy a set of somewhat contrasting requirements, which are summarized as follows:

Limited Storage Overhead. Since the number of possible queries to be posed to the system is huge and will grow over time, it is not conceivable to just do some "query book-keeping", i.e. storing the values of M_{opt} for already-seen queries. The method we seek should have a complexity *independent* of the number of processed queries and only a low (e.g. linear) complexity in the dimensionalities of the feature space and of the parameter space.

Prediction. The method should also be able to provide reasonable "guesses" even for new queries. It is also requested that the quality of this approximation has to increase over time, as more and more queries and user-feedback information are processed.

Dynamicity. Since we consider an interactive retrieval scenario, it is absolutely necessary that the method is able to efficiently handle updates, i.e. incorporate additional data without rebuilding the approximation of M_{opt} from scratch.

We have been able to achieve a satisfactory trade-off, thus meeting all above requirements, by implementing FeedbackBypass using a wavelet-based data structure, which we call the *simplex tree*.

4 Wavelets, Lifting, and Interpolation

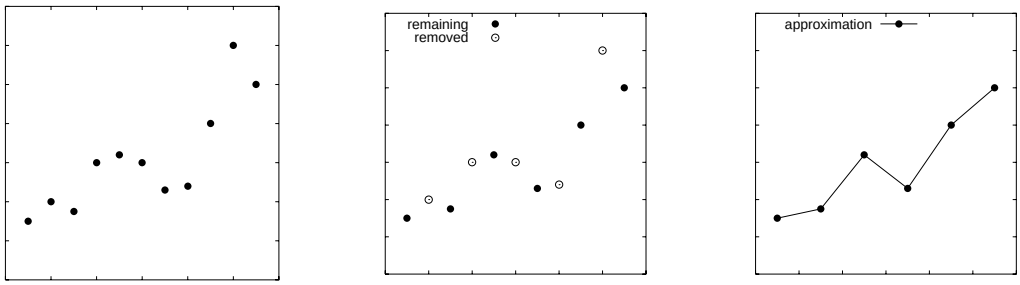
The process of *learning* a function can be understood as *approximating* the function. From the rich mathematical toolkit of approximation theory, we chose to go with wavelets constructed by a technique called *Lifting*. In this section, we briefly outline the principles but refer the interested reader to e.g. [Swe96, SS96].

The lifting schema, introduced by Sweldens [Swe96], is a highly effective yet simple technique to derive a wavelet decomposition for a given data set.

Lifting consists of three steps: *split*, *predict*, and *update*, which are repeatedly applied to the data set. Before we go into detail, it may be helpful to give the reader an intuition of the process. Essentially, the idea behind lifting is to gradually remove data from the original signal and replace it with information that allows to reconstruct the original data. This removal process is recursively repeated. At the end of each such iteration we obtain a coarser approximation of the original data and information necessary to revert the last approximation step. Finally, after the recursive application of this schema terminated we arrived at the coarsest approximation possible (e.g. one data point) but have also the information how to reconstruct the original data step by step. This proceeding, formally speaking, implements the wavelet transform.

For simplicity, let's assume the original data, usually referred to as signal, is given as pairs (x_i, y_i). Moreover, let x_i be equidistant (see Fig. 5a)). The three steps in detail are as follows:

1. In the *split* step, the original data set $Y = \{y_1, y_2, \ldots, y_n\}$ is devided into two subsets Y_1 and Y_2. Although there are no further requirements as to how to choose the subsets, let's assume



(a) (b) (c)

Figure 5: Lifting: (a) original data set, (b) split and removing of data set, and (c) predicted data by interpolation

$Y_1 = \{y_1, \ldots, y_{2k+1}\}$ and $Y_2 = \{y_2, \ldots, y_{2k+2}\}$, see also Fig 5b). We then remove Y_2 from the dataset.

2. In the next step, we *predict* each of the removed points in Y_2 by interpolating on l of the remaining neighbors in Y_1—in the case of linear interpolation, we simply use the line segment $(x_m, y_m), (x_{m+2}, y_{m+2})$ to approximate (x_{m+1}, y_{m+1}). Let $(\hat{x}_{m+1}, \hat{y}_{m+1})$ be the result of the interpolation. In the case where the interpolation coincides with the original data point, we obviously did not loose any information. However, in general, the points do not coincide. To make up for the loss of information, we determine the difference δ between the predicted and the actual value and store it in place of the original data. At the end of this step we can encode the signal using only Y_1 and $\Delta = \{\delta_1, \ldots, \delta_{k+1}\}$ (cf. Fig. 5 c)).

   l, the number of neighbor considered during the interpolation determines the degree of the polynomial function. $l = 2$ corresponds to linear, $l = 4$ to cubic interpolation etc. Special care has to be taken at the fringes of the data set though [Swe96]. In case of $l = 2$, we obtain the well-known Haar-Wavelet (see e.g. [Kai94]).

3. In the *update* step, the remaining original data is adjusted to preserve the total energy of the signal. To see what we mean by this, consider the following example where $y_{2i+1} = 0$ and $y_{2i+2} = 1$. The average value of the signal is 0.5. However after removing all values y_{2i+2}, the signal's average drops to 0, no matter the difference data we stored. Like the predict step, the update step can be performed locally by taking only a data point and it's removed neighbor into account.

Conversely, Lifting can be used to interpolate signals where data points are added successively. To do so, we simply need to reverse step 1 and 2. There is no need to apply step 3, the update step as

we do not know the total energy of the signal in advance—as a result, the approximation may change fundamentally as the dataset changes, in other words, shape and quality of the approximation are evolving with the dataset [SS96]. Like with the original Lifting technique, we may use polynoms of any degree.

When detailing the three steps above, we assumed the data to be equidistant; this assumption is valid in classical areas of application like signal or image processing. However, in the case of interpolation of a growing dataset, the x_i cannot be equidistant due to the fact that we introduce more and more points as we go. Thus, the interpolation has to take into account that intervals between data points may vary. In case of linear interpolation, we obtain of the *unbalanced* Haar-Wavelet.

5 The Simplex Tree

The *Simplex Tree* is a data structure which forms the core of our approach. Recall that we want to approximate the optimal query mapping

$$M_{opt} : \Re^D \rightarrow \Re^n, \qquad n = D + P$$

given a small but evolving sample of data points, namely queries for which feedback data is available. We can simplify the problem by decomposing this function into a set of n independent functions

$$m_i : \Re^D \rightarrow \Re$$

and express $M_{opt}(\mathbf{x})$ as $(m_1(\mathbf{x}), m_2(\mathbf{x}), \ldots, m_n(\mathbf{x}))$. Hence, we will develop the idea of Feedback-Bypass using the simple form only and discuss potential issues when integrating the decomposition at the end of this section.

In order to approximate the m_i scalar mapping using Lifting, we firstly need to organize the original, high-dimensional vector space $\Re^D$ as a collection of intervals Λ with $\bigcup \Lambda = \Re^D$. The points $\mathbf{x}$ at which feedback data $m_i(\mathbf{x})$ is available are the delimiters of the intervals. Secondly, for every new query, we have to determine the interval that contains the new query point and all its delimiters. And finally, for all feedback data for which we decide to incorporate it in our approximation, we need to update the organization of the vector space. It is highly desirable that this update affects only the interval which encloses the query point to avoid cascading effects which would make re-organization prohibitively expensive.

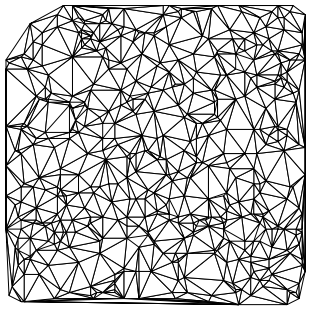


Figure 6: Example of triangulation for $D = 2$

5.1 Triangulation

Given an initial set of feedback data, we define suitable intervals on which we can base our wavelet by triangulating the set. In general, a triangulation is a decomposition into simplices, i.e. intervals spanned by $D + 1$ points—that is, triangles in $\Re^2$, tetrahedrons in $\Re^3$, and so forth. Figure 6 shows an example for $D = 2$. Triangulations are one of the fundamental problems in computational geometry and very efficient techniques to find "good" triangulations are known for low dimensional spaces [Meh84, PS85]. Computing triangulations like the Delaunay triangulation which minimizes the lengths of edges of the simplices is computational expensive and, as we found in own preliminary test, too time consuming for dimension higher than 10.

Instead, to keep the computational effort low, we use an *incremental* triangulation technique as we go forward and *split* for every new data point its enclosing simplex. More formally, let $S = \{\mathbf{v}_1, \ldots, \mathbf{v}_{D+1}\}$ be the set of points, which spans the simplex that encloses the new data point $\mathbf{x}$. Then,

$$S_i = \{\mathbf{v}_j | j \neq i\} \cup \{\mathbf{x}\}, \qquad 1 \leq i \leq D + 1$$

is a decomposition of S. Figure 7 shows examples for splits in two and three dimensions respectively. Splitting a simplex is in $O(1)$, since the dimensionality is a constant for a given data set. The space requirements of the triangulation, i.e. the number of simplices scales linearly with the number of splits g, thus is in $O(g)$.

Obviously, we can only split a simplex if the new point is inside the simplex. Hence, we have to take care that the initial simplex, consisting of artificial points, covers the entire subspace of $\Re^D$ in which our feedback data is embedded. With scaling all dimensions to fit the unit cube I^D we choose the initial simplex to be $\mathbf{B} \cup (0, \ldots, 0)$, with $\mathbf{B} = \{\alpha \cdot \mathbf{b}\}, \alpha \geq \sqrt{D}$, and $\mathbf{b}_i$ being an orthonormal basis of $\Re^D$.

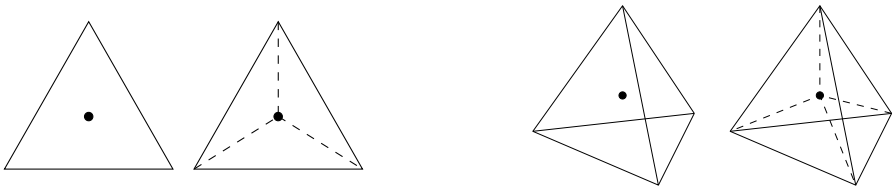


Figure 7: Splitting of simplices for $D = 2$ and $D = 3$

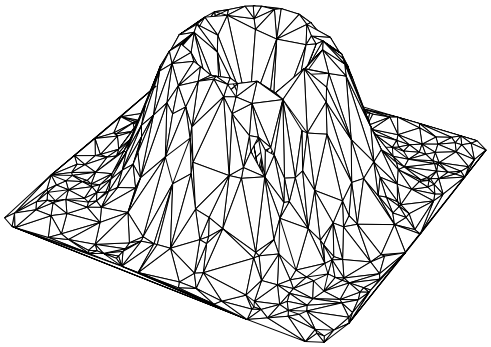


Figure 8: Example for approximation for $D = 2$

On a technical note, though $\alpha \geq \sqrt{D}$ ensures the coverage of the unit cube, greater values for α are favorable as they cause larger angles in later splits.

Triangulating the data set imposes some sort of mesh, i.e. unbalanced intervals on which we can base in the following our wavelet as outlined in the previous section.

5.2 Lookups

For each split operation we need to determine in which simplex the new point is contained. To avoid that the costs to lookup the enclosing simplex for a given new query point become dominating with an increasing number of simplices, we organize the simplices as a tree structure, hence the name *Simplex Tree*. To that end, we do not discard the original simplex when splitting it but merely add the new simplices as its children. This immediately provides for a tree index structure of degree $D + 1$ with the initial simplex as root.

We do not re-organize the tree in case it gets unbalanced due to the distribution of the data. Hence, the depth of the tree is $O(g)$, g being the number of splits, in the worst, and $O(\log_D(g))$ in the best case. We will assess the average behaviour experimentally in the next section.

5.3 Interpolation

So far we only dicussed how to organize the original feature space $\Re^D$ in intervals. To add the wavelet to the Simplex Tree, we annotate each of the points $\mathbf{v}$ in the Simplex Tree with the value $w = m_i(\mathbf{v})$, i.e. the value obtained at the end of the feedback loop which started with the query point $\mathbf{v}$. For a new query point $\mathbf{x}$, using the unbalanced Haar-Wavelet for approximating with $\widehat{w}$ the optimal query mapping value $w = m_i(\mathbf{x})$ means to perform a linear interpolation in $\mathbf{x}$.

Since a simplex, $S = \{\mathbf{v}_1, \ldots, \mathbf{v}_{D+1}\}$, by itself is a linear subspace of dimension D, we need only to solve the following equation for $\widehat{w}$

$$\begin{vmatrix} x_1 - v_{1,1} & x_2 - v_{1,2} & \ldots & x_D - v_{1,D} & \widehat{w} - w_1 \\ v_{2,1} - v_{1,1} & v_{2,2} - v_{1,2} & \ldots & v_{2,D} - v_{1,D} & w_2 - w_1 \\ \ldots & \ldots & & \ldots & \ldots \\ v_{D+1,1} - v_{1,1} & v_{D+1,2} - v_{1,2} & \ldots & v_{D+1,D} - v_{1,D} & w_{D+1} - w_1 \end{vmatrix} = 0$$

Figure 8 shows the resulting approximation by linear approximation for a synthetic example. The approximation was done using the triangulation of Figure 6.

5.4 Inserts

As opposed to typical spatial index structure the Simplex Tree is not an index which stores primarily points. Instead, it stores points to organize the feature space into simplices. We do not need to insert every point but only points which contribute to the approximation *significantly*, i.e. points for which

$$|m_i(\mathbf{x}) - \widehat{w}| > \epsilon$$

for a given threshold ϵ. In other words, if $\widehat{w}$ is almost equal to $m_i(\mathbf{x})$ there is no need to store $\mathbf{x}$ in the Simplex Tree. The particular choice of the threshold ϵ determines the quality of the approximation: for low thresholds the approximation is more accurate—with a threshold of zero every point which wasn't precisely predicted gets inserted—whereas high thresholds cause more slack. More important, however, is the character of the query mapping function. If M_{opt} is composed of low frequencies, only very little feedback data is inserted, for a query mapping composed of high frequencies, more query points are needed to reach approximations of suitable quality.

Consequently, the resource requirements of the Simplex Tree do not depend on the number of queries for which feedback is provided but on the optimal query mapping and the insert threshold.

5.5 Dealing with Numeric Instability

Depending on the data distribution and the optimal query mapping, we might end up inserting points close to previously inserted ones, or points close to a subsimplex, i.e. a facet, of the enclosing simplex. This in turn may incur numeric instability, e.g. due to numeric imprecision we cannot determine in which leaf of the Simplex Tree a point actually is enclosed.

This kind of numeric instability is inherent to multi-dimensional computational geometry. Essentially, there are two possible ways to deal with it:

- using libraries which implement infinite precision arithmetic
- rigorous checks for numerical errors and aborting of any arithmetic operation if the numerical error can no longer be assessed conclusively, i.e. the impact of the error is no longer controlable

The first possibility clearly is the preferred solution but due to its enormous costs in terms of running time, it is no alternative in our case.

Thus, we chose to implement the less expensive solution. In case of numerical problems during an insert, we simply abort the operation, i.e. the actually valuable data point is ignored. During the prediction, aborting is somewhat unsiatisfactory as no appropriate value can be returned for processing the associated query. However, the hierarchical structure of the Simplex Tree provides for a simple, yet robust solution: In case of numeric instability where we cannot determine the child simplex which encloses the query point, we fall back use the parent simplex for the interpolation.

Though depending on the data set and M_{opt}, we rarely encountered numeric problems in our experiments, and it has been our experience that their impact, when dealt with as we just outlined, is not appreciable in general.

5.6 Vector Valued Query Mappings

We simplified the original problem focusing on the approximation of only one single parameter, i.e. m was a function into $\Re$.

To handle the general case $M_{opt} : \Re^D \rightarrow \Re^n$ we maintain vectors at each point in the Simplex Tree. When interpolating we treat the parameters, however, individually and solve the above determinant equation n times, adjusting the last column accordingly. We insert a point $\mathbf{x}$ if

$$\max_i |m_i(\mathbf{x}) - \widehat{w}_i| > \epsilon$$

The other operations, lookup and split, are unaffected by the dimensionality of the optimal query mapping.

6 Experimental Evaluation

We have implemented FeedbackBypass in C++ under Linux, and tested its performance in order to answer the following basic questions:

- Which are the actual prediction capabilities of FeedbackBypass? How much feedback information does FeedbackBypass need to perform reasonably well? How long does it take to learn the optimal query mapping?
- How much do the predictions of FeedbackBypass depend on the specific data set? Alternatively, is FeedbackBypass robust to changes in the type of queries to be learned?
- How much do we gain, in terms of efficiency, by "skipping" the feedback loop?
- Which is the relationship between the storage overhead and the accuracy of prediction? Does FeedbackBypass effectively exploit storage resources?

For evaluation purposes we used the IMSI data set consisting of about 10,000 color images. [1] Each image is already annotated with a *category* (such as "birds", "monuments", etc.). From each image a 32-bins color histogram was computed and used to compare images using the class of weighted Euclidean distances. We implemented both query point movement and re-weighting feeedback strategies, as described in Section 2, which means that the query mapping managed by the simplex tree is a function from $\Re^{31}$ to $\Re^{62}$ (see also Example 1 in Section 3).

The setup for the experiments with the IMSI data set was as follows. From the whole data set we selected 2,491 images belonging to 7 categories: Bird (318 images), Fish (129), Mammal (834), Blossom (189), TreeLeaf (575), Bridge (148), and Monument (298). This subset of images was then used to randomly sample queries, whereas images in other classes were just used to add further noise to the retrieval process. For each query image, any image in the same category was considered a "good" match whereas all other images were considered "bad" matches, *regardless of their color similarity*. This leads to hard conceptual queries, which however well represent what users might want to ask to an image retrieval system. Since within each category images largely differ as to color content, any query based

[1] IMSI MasterPhotos 50,000: `http://www.imsisoft.com`,

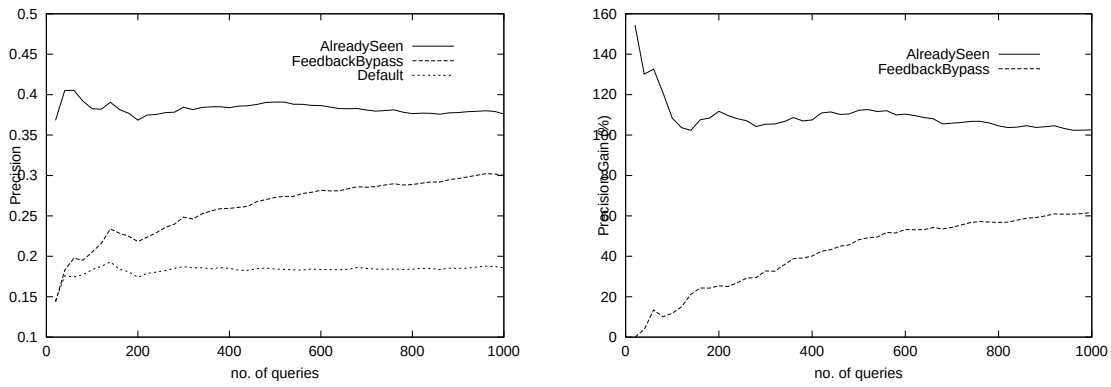
Figure 9: Sample images from the “Fish” category

on a color distance cannot be expected to find more than a fraction of relevant images to be close in color space. For instance, all the 4 images shown in Figure 9 belong to the “Fish” category: only the 2nd image (“shark”) has a dominant blue color, whereas others have strong components of yellow, gray, and orange, respectively. A similar fair evaluation procedure was also adopted in [RH00].

To measure the effectiveness of FeedbackBypass we consider classical precision and recall metrics [Sal88], averaged over the set of processed queries. For a given number k of retrieved objects, precision (Pr) is defined as the number of retrieved relevant objects over k, and recall (Re) is the number of retrieved relevant objects over the total number of relevant objects (in our case, the number of images in the category of the query).

In our experiments we used a typical value of $k = 50$, and in any case k never exceeded 80. This is because we consider that a real user will hardly provide feedback information for larger result sets. As a consequence, since the number of retrieved good matches is limited above by k (and in practice stays well below the k limit), the use of distance functions more complex than weighted Euclidean, such as Mahalanobis, was not considered. Indeed, as observed in [RH00], improvement due to feeedback information is possible only when the number of good matches is not much less than the number of parameters of the distance function to be learned, which is 31 in our case but would be $31 \times 32/2 = 496$ for the Mahalanobis distance.

The results we show for FeedbackBypass *always* refer to predictions for “new” (i.e. never seen before) queries. They are contrasted with those obtainable from the Default strategy that uses default query parameters (i.e. user query point and Euclidean distance), which is the strategy used by *all* current interactive retrieval systems. For reference purpose we also show results obtainable by our method for AlreadySeen queries, which indeed represent the most favorable situation. It can be argued that the more the results from FeedbackBypass and AlreadySeen are similar, the more FeedbackBypass is approaching the intrinsic limit established by the use of a given class of distance functions.



(a) (b)

Figure 10: Precision results: (a) absolute values; (b) gains with respect to the DEFAULT strategy

6.1 Experiments with the Image Data Set

6.1.1 Speed of learning

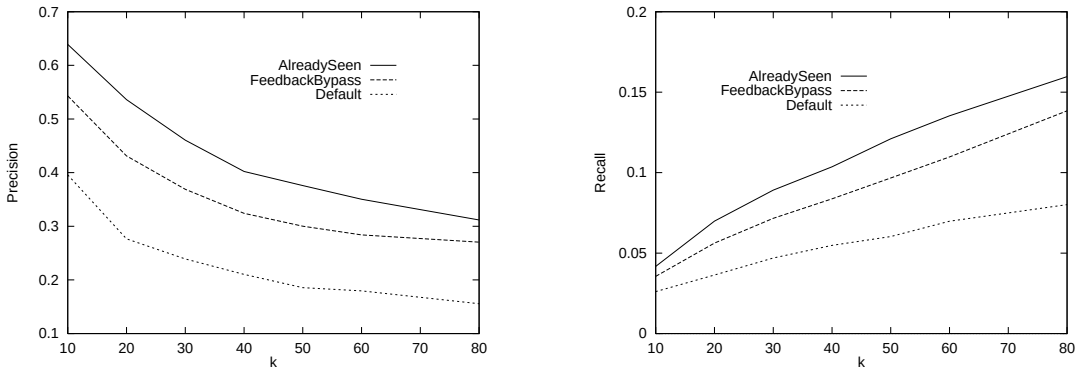
Figures 10 (a) shows average precision as a function of the number of processed queries. For this figure the number of retrieved objects was set to $k = 50$. It is evident that the performance of FeedbackBypass monotonically increases with the number of queries, and that the difference between FeedbackBypass and the Default strategy is already significant after the first few hundred queries . This is also emphasized in Figure 10 (b), where we show values of the *precision gain*, PrGain, defined as:

$$\text{PrGain(FeedbackBypass)} = \left(\frac{\text{Pr(FeedbackBypass)}}{\text{Pr(Default)}} - 1 \right) \times 100$$

and similarly for the AlreadySeen case. The number of good matches doubles for already seen queries, and increase by 60% for queries never seen before.

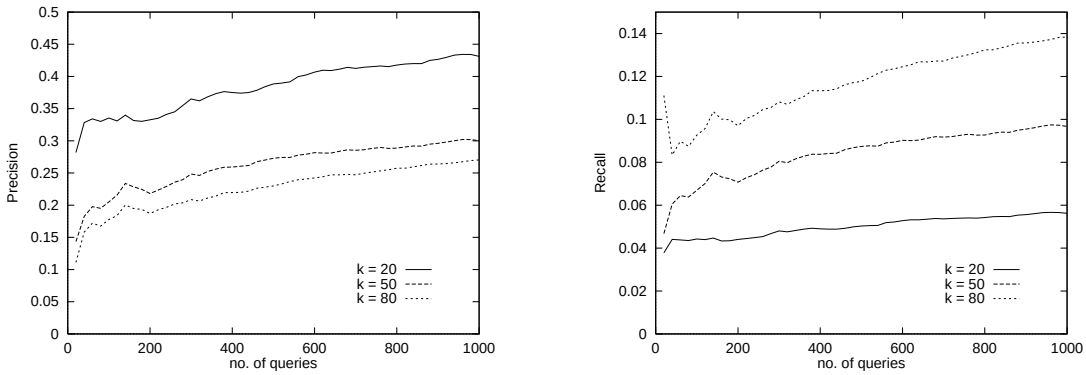
Figures 11 (a) and (b) show, respectively, the values of average precision and recall after 1000 queries, when k varies between 10 and 80. The graphs confirm that our method is able to provide accurate predictions even when the number of retrieved objects per query, k, is low. This can also be appreciated in Figures 12 (a) and (b), where precision and recall curves for $k = 20, 50$, and 80 are plotted versus the number of queries.

In the previous experiments we have considered a same value of k both to train the system and to evaluate it. However, it is also important to understand if training FeedbackBypass with larger values of k can be better than training FeedbackBypass with less information. Clearly, precision results shown in Figure 12 (a) are of little use to this pupose, since they are obtained with a different number of



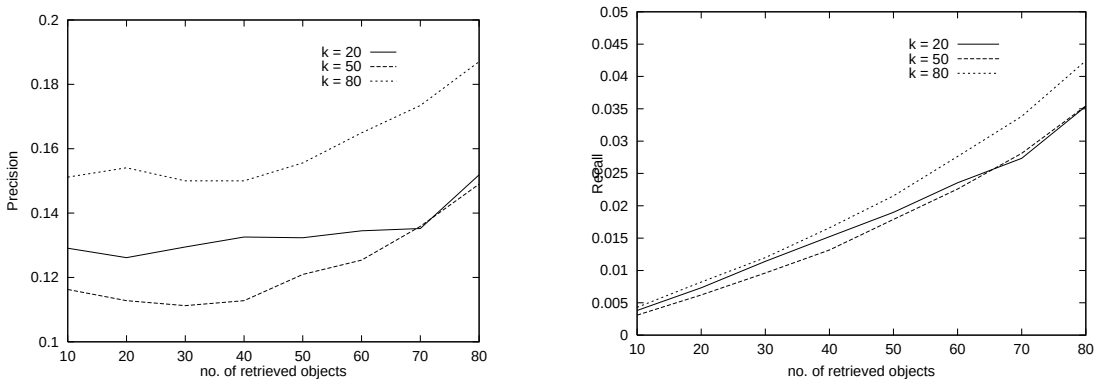
(a) (b)

Figure 11: Precision (a) and recall (b) after 1000 queries



(a) (b)

Figure 12: Precision (a) and recall (b) of FeedbackBypass for different values of k



(a) (b)

Figure 13: Precision (a) and recall (b) of FeedbackBypass for several values of k as a function of the number of retrieved objects

retrieved objects for each curve. Thus, we have compared several versions of FeedbackBypass each trained with a specific k value, when they are used to answer queries requesting the same number of objects from each version. The basic conclusion that can be drawn from the results shown in Figure 13 is that using larger k values is worthwhile, even if less objects are retrieved. This is particularly evident for the $k = 80$ curve, while less for the case $k = 50$. We argue that this is related to the intrinsic geometry of the underlying feature space, as also Figure 12 suggests (there, the difference between the $k = 80$ and $k = 50$ curves is much larger than the one between $k = 50$ and $k = 20$).

6.1.2 Robustness

We now turn to consider how much the performance of FeedbackBypass depends on the specific queries for which predictions are required. For this experiment we separately measured precision for the 7 query categories. Looking at precision results (see Figure 14 (a)) it can be observed that FeedbackBypass is able to provide useful predictions in all cases where there is a significant difference between the Default and the AlreadySeen cases. Indeed, such a difference is a clear indication that feedback information actually leads to improve the results. This is particularly evident for the largest query category ("Mammal"). On the other hand, when feedback only slightly improves the quality of the results (see the "TreeLeaf" category, denoted simply as "Leaf" in the figure), predictions for new queries do not provide benefits, as it could have been expected. This general behavior is only violated for the "Fish" category, where it seems that no improvement can be obtained from FeedbackBypass on new queries, even if performance of AlreadySeen is particularly good. However, since "Fish" is the smallest category (129 images), it can be argued that the number of sampled queries is still not enough to well approximate

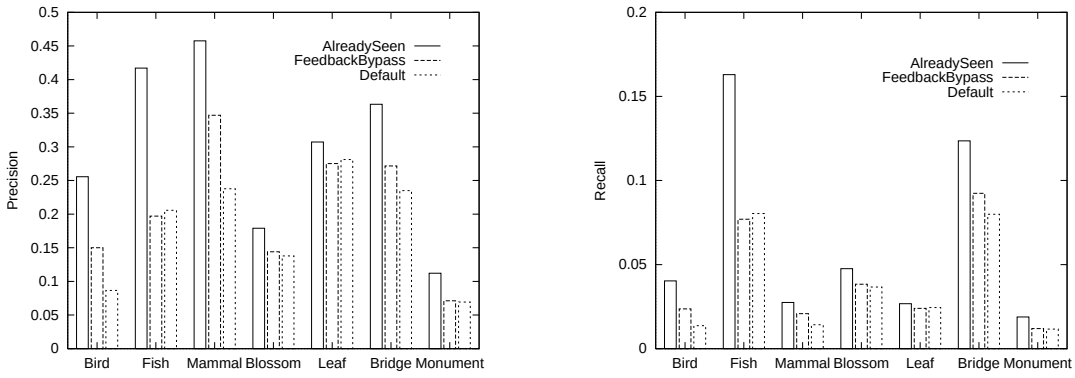


Figure 14: Precision (a) and recall (b) for the 7 query categories

the optimal query mapping for that category. Similar results are observed in Figure 14 (b) for the recall metric.

6.1.3 Efficiency

An important aspect that we analyze here is how much we can gain by using FeedbackBypass in terms of *efficiency*. Clearly, the overall performance of an interactive retrieval system will also depend on the specific access methods that are used to retrieve the stored objects, as well as by the indexed features. In order to provide unbiased results, we consider the following performance metrics:

- The average number of feedback iterations that FeedbackBypass saves with respect to the Default strategy, in order to obtain the same level of precision. Thus, for each query we start the feedback loop either from default or from predicted query parameters, and measure how many iterations are needed before no further improvements are possible. This "Saved-Cycles" measure tells us how many query requests to the underlying system we save, on the average, for each user query.

- The average number of objects that we *do not* have to retrieve for achieving the same level of precision than Default. Note that this "Saved-Objects" metric is simply computed as:

$$\text{Saved-Objects} = \text{Saved-Cycles} \times k$$

Figure 15 presents results for $k = 20$ and $k = 50$. In both cases it can be seen that the savings improve over time, and that after 1000 queries they amount to about 2 cycles for $k = 50$, which translates

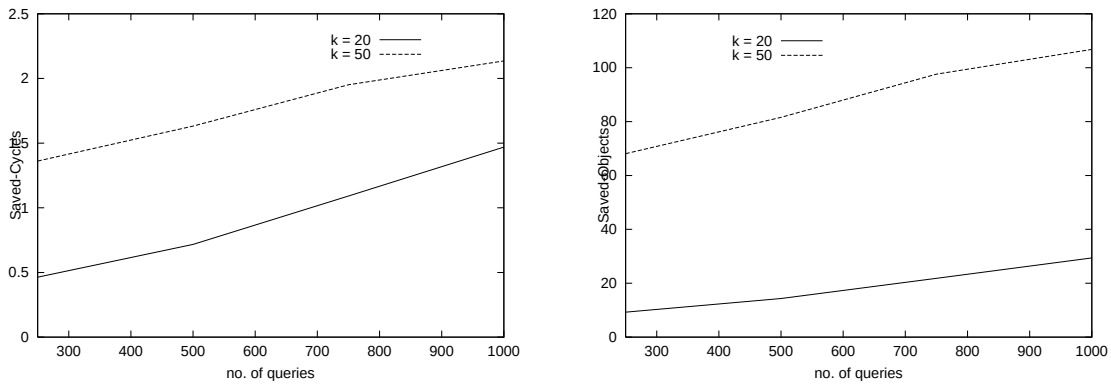


Figure 15: Average number of feedback cycles (a) and retrieved objects (b) saved by FeedbackBypass

in a net reduction of 100 objects retrieved from the underlying system.

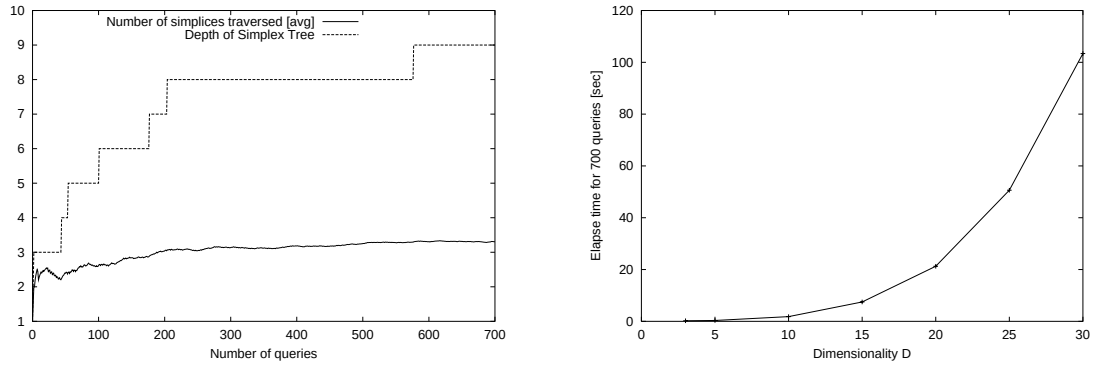
Finally, in the last experiments we want to assess the data structure as such. Figure 16a) shows the number of simplices traversed for a query session with the same settings used above. For orientation, we also plotted the depth, i.e. the maximum number of simplices that could be traversed. Both are logarithmically increasing, however, the number of simplices traversed on average and thus the elapse time per query is significantly lower than the depth underlining the efficiency of FeedbackBypass.

In Figure 16b) the elapsed time for processing a batch of 700 queries is shown as a function of the dimensionality. Recall, all operations in the Simplex Tree are either linear or logarithmic in the number of queries for a fixed dimensionality. However, when varying the dimensionality we obtain quadratic complexity as both the number of simplices and the number of arithmetic operations per simplex scale lineary with the dimensionality, thus $O(D^2)$.

7 Conclusions

In this paper we have presented the FeedbackBypass, a new method to speed-up the process of interactively searching for relevant information in multimedia databases. The key idea of FeedbackBypass is to organize the information gathered from user interaction in a multi-dimensional wavelet. Approximations obtained from this wavelet can be used to either "bypass" the feedback loop completely for already-seen queries, or to "predict" near-optimal parameters for new queries. We detailed the operations on the FeedbackBypass including inserts, lookups, and interpolation.

Our experiments show that FeedbackBypass works well on real high-dimensional data, and that its predictions consistently outperform basic retrieval strategies which start with default query parameters.



(a) (b)

Figure 16: Average number of simplices traverse per query (a) and elapsed time for query batches with variable dimensionality (b)

We have also quantified the savings FeedbackBypass provides in terms of number of queries and of retrieved objects.

A key feature of FeedbackBypass is its orthogonality to existing feedback models, i.e. the FeedbackBypass approach can be easily incorporated into current retrieval systems regardless of the particular mathematical model underlying the feedback loop. The FeedbackBypass is distinguished by its low resource requirements which grow polynomially with the dimensionality of the data set, thus making it applicable to high-dimensional feature spaces.

References

[BKK96] S. Berchtold, D.A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 28–39, Mumbai (Bombay), India, September 1996.

[CPZ97] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 426–435, Athens, Greece, August 1997.

[Fal96] C. Faloutsos. *Searching Multimedia Databases by Content*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1996.

[ISF98] Y. Ishikawa, R. Subramanya, and C. Faloutsos. MindReader: Querying Databases Through Multiple Examples. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 218–227, New York, NY, USA, August 1998.

[Kai94] G. Kaiser. *A Friendly Guide to Wavelets*. Birkhäuser, Boston, Basel, Berlin, 1994.

[Meh84] K. Mehlhorn. *Data Structures and Algorithms Vol. 3: Muti-dimensional Searching and Computational Geometry*. Springer-Verlag, Berlin, New York, etc., 1984.

[ORC+97] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. Huang. Supporting similarity queries in MARS. In *Proc. of the Int.'l Conference on Multimedia*, pages 403–413, Seattle, WA, USA, November 1997.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, Berlin, New York, etc., 1985.

[RH00] Y. Rui and T. S. Huang. Optimizing Learning in Image Retrieval. In *Proc. of IEEE Int'l. Conf. on Computer Vision and Pattern Recognition*, Hilton Head, SC, USA, June 2000.

[RHOM98] Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra. Relevance Feedback: A Power Tool in Interactive Content-Based Image Retrieval. *IEEE Trans. on Circuits and Systems for Video Technology*, 8(5):644–655, September 1998.

[Sal88] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1988.

[SK97] T. Seidl and H.-P. Kriegel. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 506–515, Athens, Greece, August 1997.

[SS96] W. Sweldens and P. Schröder. Building Your own Wavelets at Home. In *Wavelets in Computer Graphics*, pages 15–87. ACM SIGGRAPH, 1996.

[Swe96] W. Sweldens. The Lifting Scheme: A Custom-design Construction of Biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.