Efficient Video Sequence Retrieval in Large Repositories

Hari Sundaram and Shih-Fu Chang
Dept. of Electrical Engineering, Columbia University,
New York New York 10027.
Email:{sundaram,sfchang}@ctr.columbia.edu

Abstract

This paper presents algorithms to deal with problems associated with indexing high-dimensional feature vectors that characterize video data. Indexing high dimensional vectors is well known to be computationally expensive. Our solution is to optimally split the high dimensional vector into a few low dimensional feature vectors and querying the system for each feature vector. This involves solving an important sub-problem: developing a model of retrieval that enables us to query the system efficiently. Once we formulate the retrieval problem in terms of a retrieval model, we present an optimality criterion to maximize the number of results using this model. The criterion is based on a novel idea of using the underlying probability distribution of the feature vectors. A branch-and-prune strategy optimized per each query, is developed. This uses the set of features derived from the optimality criterion. Our results show that the algorithm performs well, giving a speedup of a factor of 25 with respect to a linear search while retaining the same level of Recall.

Keywords: Scalability, Visual Information Retrieval, Statistics, Shape and Motion Representations, Recall.

1. Introduction

In this paper, we address the problem of efficiently indexing very large video databases. In particular, we focus on developing algorithms that efficiently index videos using low-level features extracted from the videos. There are several reasons why this problem is important. (a) It is becoming increasingly common to find large image and video databases on the web, much of which is unannotated. Since associated text is scarce (or subjective, when available) we would like the video descriptors to be based on the low-level features extracted from the video. (b) Traditional feature-based methods to index images/video are unable to efficiently scale with the size of the database due to the computational cost of indexing high-dimensional feature vectors that characterize video data.

The effort to retrieve images using visual features (e.g. color histograms) derived from the images is of recent origin. Notable image retrieval systems include the QBIC project [9], PhotoBook [17], VisualSEEk [20] and MARS [16]. Most of the research in this area of Content-Based Visual Queries (CBVQ) has focussed on improving retrieval performance, primarily by developing new image/video features to index the data. Because of the focus on the retrieval methods, most researchers tend to work with small databases, of the order of thousands at most. On small databases, one can perform a linear search on the database, and still obtain "real-time" results.

Scaling the search to large video databases using existing methods does not work for a very good reason. Typically, the feature vectors derived from the video data are very high-dimensional. A Multifeature query may include many high-dimensional features such as motion, shape, color and texture. Indexing high-dimensional vectors is an open problem in the theoretical computer science community [1] [2] [4] [14] [19]. The cost of indexing multi-dimensional data is exponential in the dimension of the data, and beyond dimensions in the range of 12-15 it is more efficient to do a linear search [10] [24]. Some of the techniques used in the CBVQ community to partially overcome this problem include prefiltering techniques [12], variants on multidimensional scaling [8] and use of the Karhunen-Loève transform. Note that even if low-dimensional representations were to be used for each feature, the resultant dimensionality of the combined weighted feature would still be large, ruling out a fast search. We shall revisit these issues in Section 2.

An important aspect of the retrieval problem is determining the model to be used for retrieving the data. The retrieval model determines how the query is processed by the system. This problem has received some attention in the image indexing community [7] [16]. In [16], different retrieval models (Boolean, Fuzzy, Probabilistic) developed in the information retrieval community were applied to image indexing. Most of the current video/image indexing schemes use a weighted combination of features in order to compare the query (e.g. a sketch [5], an example image/video) with the videos (images) in the database. The search is linear over the database of images; this is in effect a linear search in a very large dimensional feature space.

Since indexing high-dimensional features is computationally expensive, we propose the following approach to indexing video features. (a) Only use low-dimensional representations for each feature. This will allow us to search for each individual feature efficiently. (b) Develop a retrieval model based on the Boolean AND retrieval model. This model allows us to query for each feature independently and efficiently. Now, given a user defined query and the number of videos to retrieve, pick an optimal subset of the features specified in the query. Then we use our algorithm to query the system using these optimal features and then present the results to the user. In this paper, we show the following results:

- We show that there exist some pitfalls if the AND model is chosen as the retrieval model. We then present an optimality criterion that overcomes these problems. Given the query vectors and the N_o the number of videos to retrieve, the criterion determines the optimal feature and determines the subset of the feature that maximizes the expected number of relevant videos. The optimality criterion is based on a novel idea of estimating conditional probability distributions for every pair of features.
- A branch and prune strategy that uses the criterion helps pick the optimal sub-set from the query vector. Then this subset is used to query the system and merge the results.

Our algorithms have been implemented within the VideoQ [5] framework, a sketch based video search system. VideoQ is an object based system, indexing on object level features generated from automatic processing of videos. Each object is associated with a feature vector having five component features: color, texture, shape, size and motion.

The remainder of the paper is organized as follows: In section 2, we present some background on the problems associated with high-dimensional indexing. In section 3, we discuss different retrieval models and choose one that allows us to make efficient queries. In section 4 we formulate the problem in terms of the retrieval model and present algorithms to pick an optimal feature. In section 5 we present an algorithm for retrieval based on our optimality selection of features. In section 6, we present our results and we follow that section with our conclusions.

2. Scalability

The problem of efficient indexing of very high-dimensional datasets [10] [11] [19] is better known as the "curse of high-dimensionality" [24]. Typically, all indexing algorithms perform worse than a linear search on the data. This is because algorithms indexing high-dimensional data have a cost factor of $O(2^d)$ (due to backtracking) where d is the dimension of the dataset. Hence, for dimensions greater than 12-15, the indexing strategies used at present do not scale well with the size of the database. The scaling issue is further aggravated by the use of computationally expensive metrics on the data.

If the data is static then we can perform hierarchical clustering [23] to speed up the process of retrieval. This has been extensively used in the query by example paradigm. Dimensionality reduction techniques (such as the Karhunen-Loève Transforms used in QBIC) are also useful with static data. However in both techniques, the feature vectors have to be recomputed on regular basis when videos are added or removed from the repository. Otherwise, the results of a nearest-neighbor search on the low-dimensional space will be inaccurate. This is an issue in video collections that grow on a regular basis.

In [12], the authors discuss a technique to efficiently index high-dimensional color histograms. The authors project all the color histograms onto a low-dimensional sub-space. After proving that the distance in the sub-space is a *lower bound* on the actual histogram distance, the authors search in this sub-space. The results of the search are then checked with the actual distance metric to remove false positives. However, since the distance in the sub-space is a lower bound on the original distance, we have false alarms. The authors show that the false alarm rate can be reduced by increasing the dimensionality of the sub-space. The authors obtain their best results for a sub-space of dimensionality 16. However, at this dimension, high-dimensional indexing problems will set in.

There are some recent results [1], [14] that indicate that if we use approximate range queries, we can develop a polynomial time algorithm (i.e. polynomial in d, the dimensionality of the dataset). The theory [14] shows that for an ε nearest-neighbor search in an Euclidean space, the following relation holds:

$$\mathrm{d}(p,q) \le (1+\varepsilon)\,\mathrm{d}(p',q), \tag{1}$$

where d is the metric, q is the query point, p is the nearest neighbor returned by the search and p' is the true nearest neighbor. Approximate range search techniques seem promising in multimedia retrieval problems because in very large databases the (as also noted in [24]) need for efficient retrieval may overshadow the need for exact results. However, we have seen no actual implementations of these approximate range search algorithms in image/video retrieval systems.

3. Retrieval Models

Every retrieval system has a retrieval model associated with it (i.e. once we have decided upon a set of features, how do we query the system?). In this section we examine different retrieval models and pick one that enables us to execute the query efficiently[1]. We now discuss three retrieval models based on metric thresholds: AND, OR (also known as Boolean retrieval models) and the linear constraint model which is widely used in the CBVQ community. While we limit our discussion to the case of retrieving two features, the extensions to multiple features will be obvious.

Assume that we wish to retrieve images from the database using two features *x* and *y*. Each feature has a corresponding distance metric d_x and d_y respectively. Given a query *q*, weights ω_1, ω_2 and a metric threshold δ, we obtain three different result sets, one for each model:

$$\text{if } A \equiv \{p \mid d_x(p,q) \le \delta/\omega_1\},$$
$$B \equiv \{p \mid d_y(p,q) \le \delta/\omega_2\}, \text{ then}$$
$$R_{AND} = A \cap B,$$
$$R_{OR} = A \cup B,$$
$$R_{LINEAR} = \{p \mid \omega_1 d_x(p,q) + \omega_2 d_y(p,q) \le \delta\},$$

where, R refers to the retrieved set and the subscript refers to the model of retrieval. The retrieved sets are shown as the shaded regions in Figures 1(a)-(c). Now each set is an equivalence class, with members of each class satisfying a different equivalence relation. In order to retrieve the (say) first N matches for each set, we need to order the set. This we do by assigning a membership value to each element of the set. The elements of the set R_{LINEAR} have a natural membership value given by the following function:

$$\varphi(p) \equiv \omega_1 d_x(p,q) + \omega_2 d_y(p,q) \ \forall p \in R_{LINEAR} \qquad (2)$$

The sets R_{OR} and R_{AND} are unranked sets, and we could use any positive function (e.g. $\phi(p)$) to assign the membership value to the elements of these sets. Other examples of membership functions include the fuzzy [15] membership functions:

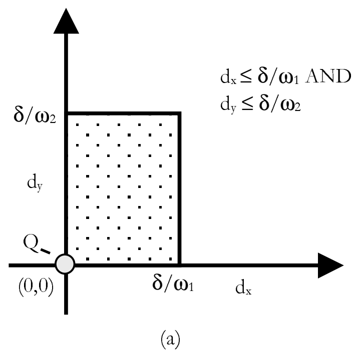
$$f_{AND}(p) = \max(d_x(p,q), d_y(p,q)),$$
$$f_{OR}(p) = \min(d_x(p,q), d_y(p,q)),$$

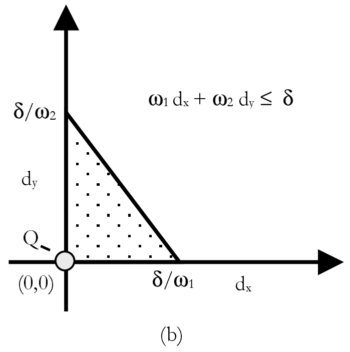
where, f_{AND} and f_{OR} are used with the AND and the OR retrieval models respectively. Now that all the three sets can be ordered according to their membership values and we can perform a rank-based retrieval on each set i.e. retrieve the top N items (this is the traditional rank query) from the set using the ordering.

In order to evaluate the retrieval effectiveness of each model, we need to use the standard information retrieval metrics of Precision and Recall (See Figure 2; Precision = c/(c+b) and Recall = c/(c+a); a, b, c refer to the size of the sets). Note that Precision and Recall need require subjective testing by user evaluation. For the purposes of retrieval, all the three models presented here are valid i.e. there is no *a priori* reason (in terms of Precision and Recall) to choose one model over the other.

We use the AND model of retrieval with the membership function being given by Equation 2. Our choice was governed by several reasons, mainly guided by computational efficiency:

- The use of the linear constraint model implies that we are in effect, searching in a high-dimensional space. This is true even if we only have five or six low-dimensional features. Using the arguments presented in Section 2, only a linear search is feasible.
- The linear constraint makes sense, if it were true that for a given semantic class, the distance of an image belonging to that class with respect to each of the features were linearly constrained. However, there is no empirical basis for putting a linear constraint on the features.
- In the AND model, the search for each feature is independent of the other. Hence we can execute each search in parallel. Since we *only* use low-dimensional





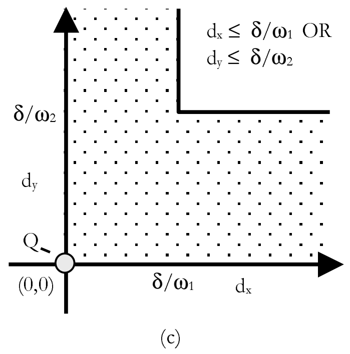


Figure 1: Three retrieval models: (a) AND (b) Linear constraint (c) OR. d_x and d_y are the metrics for features x and y respectively. Q: query, ω_1 and ω_2 are the weights.

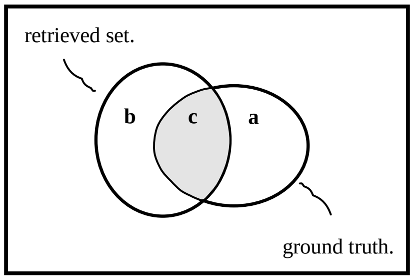


Figure 2: Defining Precision and Recall

[1] Here, the term efficiency is used in the sense of computational cost of retrieval.

feature representations in our system, the search for each feature is computationally efficient.

- Note that for a given δ and ω_i's, the set retrieved using the linear constraint model is a subset of the set retrieved using the AND model (see Figures 1(a)-(c)) i.e $R_{LINEAR} \subset R_{AND}$. We can determine R_{LINEAR} from R_{AND} by imposing the linear constraint on R_{AND}.

Note that the OR query can be done by using the AND model. Our choice of the membership function (we use Equation 2) was *ad-hoc,* we could have chosen fuzzy membership functions.

4. Optimal Feature Selection

Now that we have decided upon our retrieval model and the membership function, we shall now examine different strategies to for retrieving features using our model. Other retrieval models make either implicit or explicit assumptions (e.g. [16]) that the different features are independent. Instead, in our retrieval model, we exploit the correlations that exist in real-world data.

In Section 4.1 we reformulate the visual retrieval problem in terms of our model. In Section 4.2 we illustrate some of the problems with naïvely using the AND retrieval model. There, we also present an optimality criterion to overcome the problem. In Section 4.3 we discuss a method to generate some baseline statistics, while in Section 4.4 we present a technique to compute the optimal set using the conditional probabilities on the features. In Section 4.5 we present an assumption that simplifies our solution.

4.1 Problem Formulation

Given a query with k features $f_1, f_2 .. f_k$, the desired set D_k associated with each feature is defined as follows:

$$D_k = \{ p \mid \mathrm{d_k}(p,q) \le \delta(\omega_k) \}, \qquad (3)$$

where, q, is the query, d_k is the corresponding metric and $\delta(\omega_k)$ is a metric threshold which is a function of the feature weight ω_k (See Appendix A for the function used in this paper). The target set S under our retrieval model is then:

$$S \equiv \bigcap_{i=1}^{i=k} D_i. \qquad (4)$$

Now, in CBVQ systems, the user is interested in looking only at the top few return results, perhaps as few as 16. However, each of the sets D_i and the sets S can be very large. Now, we define the retrieval problem in terms of our model:

Problem Statement: Given a query Q with k features $f_1, f_2 .. f_k$, and N_o, the number of results wanted by the user, determine a set S_0 such that:

- $E(p \in S \mid S_o) > N_o$.

Where, E( ) is the expectation operator and S is the target set (Equation 4). Assume that we know the set sizes $|D_i| \forall D_i$ and the set size $|S|$.

Solution outline:

- Determine optimal D_o where $o = \arg\max_i \{ \mathrm{P}(p \in S | D_i \}$. This is equivalent to choosing D_i with the smallest set size.
- Note that $S \subseteq D_i \; \forall$ i. $\therefore S \subseteq D_o$.
- Determine a set $R_{opt} \subseteq D_o$ such that $E(p \in S \mid R_{opt}) > N_o$. The solution set S_o is the set R_{opt}. The number of objects retrieved is $|S_o|$.

Hence, in this manner we have mapped the retrieval problem into a problem of determining R_{opt}.

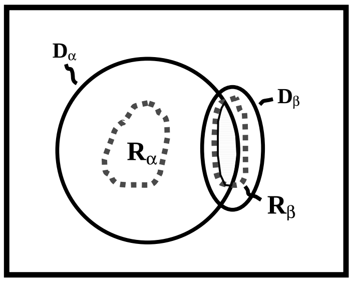


Figure 3: The dashed sets are the retrieved sets and the solid sets represent the desired sets.

4.2 The Null Set Issue

In this section, we show some of the problems with directly estimating the target set S using Equation 4. We shall also present our solution to overcome these issues. Assume that we have a single object query with two feature attributes α, β (e.g. color and motion). The sets D_α and D_β (defined using Equation 3) represent the desired sets for features α, β respectively. Let us query the system for α, β and retrieve sets R_α and R^2_β. Consider Figure 3. The set with the solid line indicates the desired set for

[2] We shall use the term "retrieved set" for R_i.

that feature while the set with the dashed lines represents the retrieved set. Now keeping Figure 3 consider the following schemes to determine the target set S (Note that from Figure 3, S is non-null):

1. If we use both R_α and R_β, the estimate of the target set S is a null set. i.e.

   $\hat{S}_1 = R_\alpha \bigcap R_\beta = \phi$

   Here, we execute two queries, one each for α and β resulting in retrieved sets R_α and R_β. Then the target set S is the intersection of the retrieved sets.
2. If we only use to the set R_β in order to estimate the target set S, the result is a non-null set since:

   $\hat{S}_2 = R_\beta \cap D_\alpha$

   In this case, we only query for β resulting the retrieved set R_β. Then we use Equation 3 on R_β to filter out those that satisfy the criteria for D_α.
3. If we only use the set R_α to estimate the target set S, the result is again a null set:

   $\hat{S}_3 = R_\alpha \cap D_\beta = \phi$

   In this case, we only query for α resulting the retrieved set R_α. Then we use Equation 3 on R_α to filter out those that satisfy the criteria for D_β.
4. If we use both sets R_α and R_β but estimate S as follows:

   $$\begin{aligned}\hat{S}_4 &= (R_\alpha \cap D_\beta) \cup (R_\beta \cap D_\alpha) \\ &= \hat{S}_2\end{aligned}$$

   This is just the union of the results from schemes 3 and 4. However, in general, $\hat{S}_2 \subseteq \hat{S}_4 \subseteq S$.

How do we pick the optimal scheme? The answer lies with the feature probability distribution of features α and β. As is clear from Figure 3:

$$\begin{aligned}&\mathrm{P}(p \in D_\alpha \mid R_\beta) \geq \mathrm{P}(p \in D_\beta \mid R_\alpha), \\ &|D_\beta| \leq |D_\alpha|.\end{aligned} \tag{5}$$

Picking the feature with the larger conditional probability will maximize the likelihood that both features will be present in the set retrieved using that feature alone. But this simply means that given features α and β, we always pick the one with the smaller desired set size. In Section 4.4 we show a technique to determine R_{opt}, the optimal subset of D_o.

It is easy to see that the arguments presented in this section, generalize to *k* features. i.e. the optimal feature to pick is always the one with the smallest set size. This must be so since given the size of the target set $|S|$, the D_i with the smallest set size will have the highest conditional probability.

4.3 Generating Statistics from the Database

In order to make use of Equation 5, we need to estimate the probabilities dynamically, since the features and the metric threshold δ change with *every* query. Since we need to estimate the partial probabilities on the fly, we need to precompute and store some baseline probability distributions, over the entire database of objects. We do so by finely quantizing[3] each feature space that is used in the system. For example, color is quantized into 166 bins, size into 10 bins etc. (refer to [21] for more details). Then we estimate joint statistics for every pair of features, by examining all the objects in the database. This method is easily explained with a concrete example.

Consider the sub-problem of creating joint statistics for the features color and size. This would imply that we create a two dimensional array of size 166x10. Now given an object in the database, we map the color and the size attributes of that object into the appropriate bin in the 2D array and increment the count for that bin. This we do for each object in the database. Hence, at the end of this procedure, we would be able to compute probabilities such as P(color=red), P(size=3), P(color=red/size=3) and P(size=3/color=red) etc. This procedure is repeated for each pair of features. In [21], we discuss an efficient procedure for computing all the statistics by making a single pass through the database. There is point to be kept in mind: The statistics just computed are of the form $\mathrm{P}(p \in \hat{D}_\beta \mid \hat{R}_\alpha)$. In order to estimate probabilities of the form $\mathrm{P}(p \in \hat{D}_\alpha, p \in \hat{D}_\beta \mid \hat{R}_\gamma)$, we keep the issues raised in [6] in mind and use:

$$\mathrm{P}(p \in \hat{D}_\alpha, p \in \hat{D}_\beta \mid \hat{R}_\gamma) = \mathrm{P}(p \in \hat{D}_\beta \mid \hat{R}_\gamma)\mathrm{P}(p \in \hat{D}_\alpha \mid \hat{R}_\gamma). \tag{6}$$

[3] This is a scalar quantization on each dimension of each feature.

This independence assumption (that α and β are independent with respect to γ) is standard in probabilistic retrieval. This is necessary since estimating higher order conditionals from the data is difficult due to high computational and storage requirements[4].

4.4 Determining D_o, R_{opt} and S_o

Given a query with two features α, β we take the following steps:

1. Compute the metric thresholds $\{\delta_\alpha, \delta_\beta\}$ (see Appendix A).
2. To compute the optimal feature o:
   - For each feature (e.g. α), compute the quantized value. Determine the quantized desired set: $\hat{D}_\alpha = \{p \mid \mathrm{d}_\alpha(p,q) \le \delta_\alpha, p \in \alpha_q\}$, where α_q is the quantized feature space. It has two important properties.
     (a) The elements of $\hat{D}_\alpha$ are mutually exclusive i.e. $P((q \in \alpha_1) \cap (q \in \alpha_2) \mid \alpha_1, \alpha_2 \in \hat{D}_\alpha) = 0$, where q is an arbitrary point in α.
     (b) $D_\alpha \subseteq \hat{D}_\alpha$. Hence $\hat{D}_\alpha$ is an upper bound estimate of the desired set D_α. This bound is tight if the feature space is finely quantized. For the remainder of this section, we assume $\hat{D}_\alpha \cong D_\alpha$.
   - Now $\mathrm{P}(p \in \hat{D}_\alpha)$ is simply the sum of the individual probabilities of the elements of $\hat{D}_\alpha$. i.e.
   - $$\mathrm{P}(p \in \hat{D}_\alpha) = \sum_{k=1}^{L} \mathrm{P}(x_k | x_k \in \hat{D}_\alpha), \qquad (7)$$
     where L is the number of quantized elements in $\hat{D}_\alpha$.
   - Hence the optimal feature is $o = \arg\min\{\mathrm{P}(p \in \hat{D}_\alpha), \mathrm{P}(p \in \hat{D}_\beta)\}$.
3. To determine the quantized optimal set $\hat{R}_{\text{opt}}$. Assume that α was determined to be the optimal feature.
   - If $|\hat{D}_\alpha| \le N_o$, then $\hat{R}_{\text{opt}} = \hat{D}_\alpha$. Then $S_o = \hat{R}_{\text{opt}}$.
   - Else, we adopt the following procedure. Note, that using our 2D statistical table (ref. Section 4.3) we know for each $x_i \in \hat{D}_\alpha$, the size of x_i : $|x_i|$ and the number in x_i that belong to S: $|x_i| \, \mathrm{P}(p \in \hat{D}_\beta | x_i)$. Then we assign a membership value to each element x_i using the function $\mathrm{P}(p \in \hat{D}_\beta | x_i)$ to order the x_i's. Then we sort the set $\hat{D}_\alpha$ so that x_1 has the highest membership value. Now define the variables M_k and N_k as:

     $M_k \equiv \bigcup_{i=1}^{i=k} x_i$ where $x_i \in \hat{D}_\alpha$, where $k \le L$, the cardinality of $\hat{D}_\alpha$. Note, $M_L = \hat{D}_\alpha$.

     $N_k = \sum_{i=1}^{i=k} |x_i| \mathrm{P}(p \in \hat{D}_\beta | x_i)$, $x_i \in \hat{D}_\alpha$, where $k \le L$. Note that $N_L = |\hat{D}_\alpha| \, \mathrm{P}(p \in S | \hat{D}_\alpha)$.
   - Define the index l as:
     $l = \arg\min_i \{N_i > N_o\}$ where N_o is number desired by the user. Then,
   - $\hat{R}_{\text{opt}} = M_l$. Hence $S_o = \hat{R}_{\text{opt}}$. and $|S_o| = N_l$.

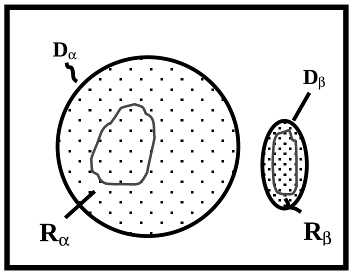


Figure 4: The case of uniform distribution of elements of S in the desired sets. The dots in D_α and D_β represent the elements of $D_\alpha \cap D_\beta$.

[4] To compute statistics of the form P(α,β,δ/γ) we would need a large four dimensional array. For the features color, size, shape and texture we would require 166 bins for color, 10 bins for size, 1000 bins for shape and 56 bins for texture. Hence, to compute P(color=red, size=4,shape=785/texture=40) we would need a datastructure of size 166x10x1000x56. Clearly, this is infeasible to implement.

4.5 A Practical Assumption

Let us now briefly examine a practical issue with the method in the previous sub-section. In order to retrieve the set S_o, we need to execute multiple queries. This point is best clarified with an example. Let $\hat{D}_\beta$ (the optimal feature selected) have 10 elements and let $\hat{R}_{\text{opt}}$ have 5 elements. Then, to retrieve S_o, we need to execute 5 queries, i.e. one query to retrieve each element of $\hat{R}_{\text{opt}}$. Formally:

$$S_o = \bigcup_{i=1}^{i=5} \{ p \mid \mathrm{d}_\beta(p,\beta_i) < \Delta_\beta \} \beta_i \in \hat{R}_{opt},$$

where, d_β is the metric for the feature β, Δ_β is the quantization threshold for the feature space β. Hence, making no assumptions on the distribution of the target set S in the sets D_α and D_β, increases the computational cost of retrieval. It also makes the retrieval scheme complex.

An assumption on the distribution of S simplifies the solution. Assume that the elements of the set S[5] are uniformly distributed in the sets D_α and D_β (see Figure 4). Now, keeping Figure 4 in mind, we see that none of the four retrieval schemes will yield a null set. With the uniform distribution assumption, the optimality criterion remains the same i.e we always pick the set with the smaller set size (Equation 5). It is easy to see that the that given the uniform distribution assumption the retrieved set size $|S_o|$ is given by:

$$|S_o| = \min\left(\frac{N_o |D_o|}{|S|}, |D_o| \right), \tag{8}$$

where D_o is the optimal feature selected. Some points need to be kept in mind about Equation 8: (a) We only query the system if $|S| \neq 0$. (b) If $N_o > |S|$, then we can only retrieve $|S|$ relevant elements. (c) If we use Equation 8, then the expected relevant set size $E(\{p \in S \mid S_o\}) = N_o$. Using technique derived in the previous section, we *guarantee* that relevant set size is greater than N_o. Due to the distribution assumption, S_o can be *any* arbitrary subset of D_o. Clearly, we can retrieve the set S_o with a single query. This can be done using the $|S_o|$ nearest neighbors of f_o, where f_o is the feature corresponding to the desired set D_o.

5. The Branch and Prune Algorithm

In this section, we present an algorithm that retrieves videos from the database given a query with *k* features. The algorithm as presented below, assumes a single object query. This algorithm makes use of the results obtained Sections 4.1-4.5. Specifically, we assume the AND retrieval model, and a membership function ϕ. The membership function (refer to Section 3) in our model is just the global distance. The membership function $\phi(q,x)$ is computed as:

$$\varphi(q,x) = \sum_{k=1}^{k=N} \omega_k \, \mathrm{d}_k(q_k, x_k), \tag{9}$$

where N is the number of features in the query ($N \leq 5$ in VideoQ), ω_k is the weight for feature *k* and d_k is the metric associated with feature *k*.

We present the discussion of our branch and prune algorithm by taking the concrete example of VideoQ. In VideoQ, we have five distinct features: color, texture, shape, size and motion. Given a VideoQ query object, we take the following steps:

1. Check the number of features in the query object. If the number of features specified is greater than three we break the set of features into two smaller sets. In this manner, we split the input query into two sub-queries. Each sub-query is executed independently of the other, in parallel. For example if five features were specified, they would be broken into two sets of size two and three features respectively. This split is done for two reasons:
   - We only have partial statistics of the form. $P(\alpha/\beta)$[6]. Computing conditionals such as $P(\alpha,\beta,\gamma,\delta/\varepsilon)$ is not feasible due large computational and storage requirements.
   - We need to break up the features into subsets containing uncorrelated features. This is done to ensure that Equation 6 holds. For example, color and texture are usually correlated (e.g. forests are usually green, etc.) and hence it may be a good heuristic to place them in different sets. At the moment, our algorithm to split the features only ensures that color and texture are placed in different sets.

[5] Note that $P(S) = P(D_\alpha \cap D_\beta)$

[6] This is notationally equivalent to $P(p \in D_\alpha | D_\beta)$.

2. If the number of features is greater than 3: There are two sub-queries (Figure 5(a)). Each sub-query contains a subset of the set of features in the original query.
   - Now, use the algorithm presented in Section 4, to determine for each sub-query, the optimal feature in the sense defined in Section 4. The algorithm also provides us with the number to retrieve for each sub-query.
   - Use the optimal feature obtained for each sub-query, to query the system. These two queries can be executed in parallel. The number to retrieve for each feature is given by Equation 8.
   - Now, we *first* take the union of the two returned sets R_{f1} and R_{f2} and then re-sort based on ϕ.
3. If the number of features is less than 3, then there is only one query (Figure 5 (b)). We again pick the optimal feature and query the system. The return set is re-sorted using the membership function ϕ (see Equation 9).

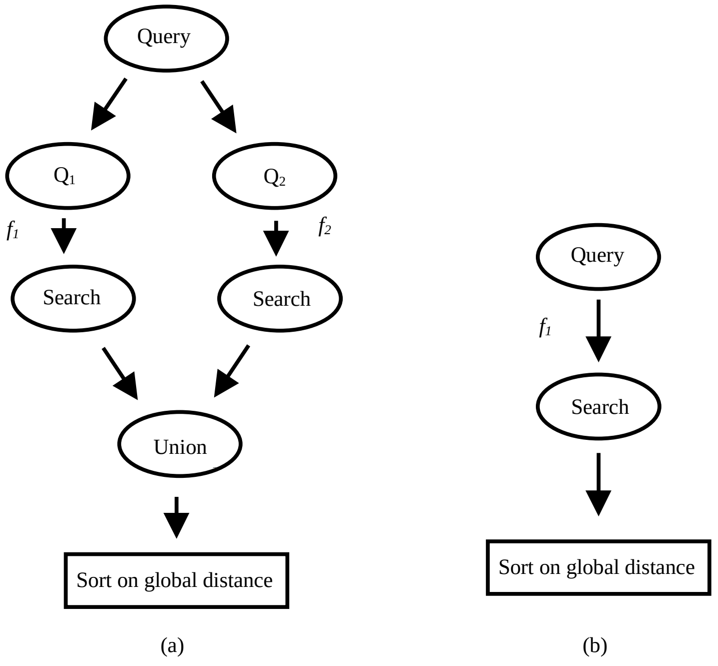


Figure 5: Branch strategies:(a) More than three features. Q1 and Q2 are sub-queries. f_1 and f_2 are the optimal features picked for sub-queries Q1 and Q2. (b) Three or less features. f_1 is the optimal feature picked.

For arbitrary number of features, we would have to split up the features into groups of three (i.e. each subset contains three features). Then as before, determine the optimal feature for each sub-query and perform a search. Then, take the union of the results of each sub-query. Again, we sort the results using the membership function ϕ. At present, we perform a multiple object query by querying for each object and then taking the set union the results of the query for each object. Then they are sorted according to a membership function.

Clearly, the procedure that we show in this section here is sub-optimal for the case of more than three features. This is (as we have noted before) due to the problem of not having accurate estimates of higher order conditionals. Note that even for the case k ($k > 3$) number of features, the optimal method involves picking the feature with the smallest desired set D_i. Hence, even in that case, one of the features picked will have been the optimal feature. However, a sub-optimal subset of the feature would have been chosen i.e. R_{opt} due to our algorithm would be sub-optimal.

6. Experimental Results and Discussion

In section 6.1 we give a brief description of the representation schemes used in our experiment. In sections 6.1.1-2 we shall give short descriptions of our low-dimensional representations for shape and motion (details in [21]). These two feature that have been traditionally hard to index. Then in Section 6.2 we provide experimental results for the retrieval scheme developed in Sections 4-5.

6.1 Representations for Retrieval

The feature set that we use builds upon the set used in the VideoQ project. All these features are attributes of video objects. We define a video object to be a contiguous set of pixels that is homogeneous (spatially as well as temporally, across frames) in the features that we are interested in (i.e. texture, color, shape, size and motion). These regions are automatically segmented and tracked over the duration of the video shot [5]. The color feature used is a triple in the LUV color space. We use the first three Tamura textures [22] for the texture feature. Size is one dimensional. Shape and motion are high-dimensional and to alleviate the problem, we use low-dimensional (implementation details in [21]) *approximations* for both of these features. All the features used in our system are low-dimensional and are hence amenable to efficient indexing schemes.

6.1.1 Shape

The shape approximations that we choose to use are widely used in the field of Geographical Information Systems (GIS) [4]. We classify the each object's shape attribute into four types: circles, squares, rectangles, ellipses. We use a metric based on geometric-shape difference to make this classification. Our shape approximation algorithm is independent of scale and leads to logarithmic indexing. We use the following shape metric:

$$\mathrm{d}(q,b) = \left|f_q - f_b\right|\omega_1 + \min\left(\left|\theta_q - \theta_b\right|, 2\pi - \left|\theta_q - \theta_b\right|\right)\omega_2 + \left|R_q - R_b\right|\omega_3 \tag{10}$$

Where *f* refers to the geometric fitting error, θ to the direction of the principal axis of the shape and *R* the aspect ratio of the shape. The ω_i*'s* refer to the different weights and the labels *q* and *b* refer to the query and the database shape respectively.

6.1.2 Motion

A motion trail describes the trajectory of the centroid of the video object. In VideoQ, the motion trail is simply a collection of points in the *x-y* plane. We decompose all motion trails [21] into a sum of linear and quadratic segments.

$$\mathrm{x(s)} = as^2 + bs + c$$

$$\mathrm{y(s)} = ds^2 + es + f$$

This level of abstraction is intuitive and is amenable to user interaction. It is also low-dimensional and easily indexible. The metric for motion trail comparison is a Mahalanobis distance:

$$\mathrm{d}(q,b) = \sqrt{\sum_{i=1}^{4}(\mathbf{t}_q - \mathbf{t}_b)\mathbf{\Lambda}^{-1}(\mathbf{t}_q - \mathbf{t}_b)^T} \tag{11}$$

Where, **t** is the four-parameter feature vector for each segment[7] and Λ is the cross-correlation matrix of the parameters; *q* and *b* refer to the query and the database trail respectively.

6.2 Evaluation Results

All our experiments took place over the video database used in our VideoQ system. The video database comprises over 2500 clips with diverse subjects such as sports, nature, travel etc. Each clip is processed giving a set of video objects associated with the clip. These objects have been obtained using the segmentation and tracking algorithms developed for VideoQ. Each video object has five attributes: {color, shape, size, texture, motion}. In our system, we need to deal with 25,000 video objects.

There are two issues that are of concern when evaluating our system. Since we address the issue of scalability, the computational cost of retrieval. And since we focus on efficient visual retrieval models, the performance of the system using standard information retrieval techniques. Hence, we choose to evaluate our system using the following metrics:

- System effort
- Precision and Recall

In our retrieval model, the computational cost of retrieval can be decomposed into three costs: indexing costs, disk access costs and the cost of computing the metric. In this paper, we ignore disk access costs. Since all our features are low dimensional, the metric cost for feature comparison can be assumed to be the same across features. The indexing cost is a function of several parameters:

1. The number of optimal features chosen, since each optimal feature must be indexed. In our implementation, this number can be either 1 or 2 (since N ≤ 5, where N is number of features in the query).
2. The user weight for each feature.

[7] We translate each segment to the origin.

3. The joint probability distribution. The weight and the probability distribution together determine the number of objects to retrieve for each feature.

Since, our algorithm picks the features used to query the system dynamically, the indexing cost[8] is a function of several random variables. Hence, to simplify analysis, we assume that the computational cost of retrieval is proportional to the number of objects retrieved from the database. This is defined to be the system effort. Had we used the linear constraint model (ref. Section 3), the cost due to the combined dimensionality of all the features would have been significant. For example even with the low-dimensional feature representations discussed in Section 6.1 the combined dimensionality of features is 18-22[9]. Clearly, the "dimensionality curse" will prevail if we attempt to use the linear constraint model to index using a structure such the R-tree or the k-d tree. Now, in order to evaluate our retrieval model we need to answer two questions:

- Does the optimality criteria really work? Instead of using Equation 5 to pick our feature, could we have selected any sub-set of query features and still obtained (or bettered) the results due to our "optimal" features?
- How does our retrieval model compare against the linear constraint model? This comparison is necessary, since most of the current CBVQ systems use this model.

In order to answer the first question, we compare the retrieval results due to the optimal set of features selected by our algorithm, against results due a random sub-set of the query features. This comparison is done using the metrics of Precision and Recall. Since the system effort is a random variable[10], we equalize the system effort in this comparison artificially, by holding it constant at 1000 (we always retrieve 500 results for each branch. i.e. set $|S_o|=500$ for each branch). This is done to prevent the variability in the system effort from affecting our results. As the database size is 25000, this also implies that we only retrieve about 4% of the database. The document cut-off value (the number of results actually shown to user), was chosen to be 16. The weights for the different features were held constant across the experiment.

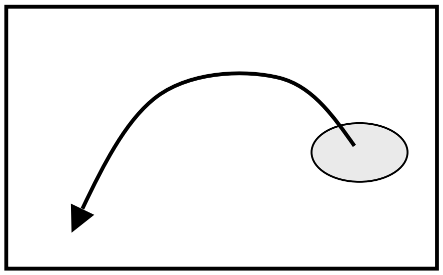


Figure 6: An example query of a high jumper.

The query used in this experiment (see Figure 6) was that of the high jumper. There are nine high-jumper video sequences in the database (out of 2500 clips). The object shown in Figure 6 has four attributes: color, motion, shape and size. The user also sets weights for each feature. Given the query, we use our branch and prune algorithm to determine the optimal set of features for this query. Since there are four features, we have two sub-queries Q_1: {color, size} and Q_2: {shape, motion}. As mentioned earlier, this split is deterministic and our splitting algorithm only ensures that color and texture are not in the same sub-query. In this case, the optimal feature set chosen was {color, motion}. In order to create queries using a random selection, we pick two features at random, from the set: {color, motion, shape, size}. Table 1 shows the results of the experiment. In the table, the "Feature Selected" column refers to the two features chosen (f_1 and f_2 in Figure 5 (a)) in order to retrieve the high jumper.

Features Selected	Precision	Recall
Our Algorithm	4/16	4/9
Color and Shape	0/16	0/9
Size and Shape	0/16	0/9
Color and Size	0/16	0/9
Size and Motion	4/16	4/9
Shape and Motion	4/16	4/9

Table 1: Precision-Recall values with constant system effort (1000) (i.e. we examine only 4% of the database.). The document cut-off value was 16.

The results in Table 1 make interesting reading. Choosing the incorrect feature set *without* the motion parameter gives very poor results. In case of the query scenarios of {color, shape} and {size, shape}, when the document cut-off value was increased to 40, Precision and Recall increased to 1/40 and 1/9 respectively. The Precision and Recall values did not change in the case of {color, size}. The reason why the Precision-Recall values were identical whenever motion was chosen is because motion is a very

[8] Since all of our features are low-dimensional and each optimal feature is queried separately under our model, the dimensionality of the feature does not contribute significantly to the indexing cost.

[9] Dimensions for each feature: color=3, size=1, texture=3, shape=3, motion= 8-12. Total: 18-22.

[10] Note, the number of objects retrieved per feature is a random variable.

strong characteristic of the class of high-jumpers. In the case of more than three features (Figure 5 (a)) we take the *union* of the results of each sub-query. Since we sort using the membership function ϕ (the global distance), the results are influenced by whether motion was selected as a feature. If motion was selected as one of the features, due to strong matches because of motion, the objects selected using the motion feature appear at the top of the list.

Now, we applied the linear constraint model to the query. Due to the high-dimensionality of the combined feature-set we had to perform a linear search on our database of objects, to obtain the retrieved set. Again using a document cut-off value of 16, we obtained the same result (i.e. the same Precision-Recall values) as the optimal case due to our algorithm. Since there are 24545 objects in our database the system effort due to a linear search is 24545. This is to be compared against a system effort of 1000 for our retrieval model.

7. Conclusions

A novel approach to indexing high-dimensional video feature vectors is presented in this paper. Given a query and the number of relevant videos desired (N_o), the proposed scheme splits the high-dimensional query feature vector into subsets of low-dimensional feature vectors. It determines an optimal feature for each subset using pair-wise conditional probabilities for the features within the subset such that the expected number in the retrieved set is greater than N_o. This procedure is *dynamic* i.e. all the parameters used and decisions made by the algorithm change with every query.

In this paper, we develop a variant of the Boolean AND retrieval model to be used in our system. We show that our retrieval model enables us to retrieve videos from the database efficiently. We also showed that a naïve use of the AND model can yield very poor retrieval results. We then presented an optimality criterion that maximizes the expected number of videos that satisfy our retrieval model. This criterion depends upon our ability to compute query-time conditional probability estimates of the features. We also presented algorithms to generate some baseline statistics on the data which are used to compute the conditional probabilities at query time.

We also presented the branch and prune algorithm which used the optimality criterion to split the input query vector and subsequently query the system. The retrieval model was evaluated using two performance criteria: the system effort and Precision-Recall. The performance results show that the model developed to enable efficient querying, works well. For the case of the high-jumper class, it yields the same results as the linear constraint model. It is important to note that this was achieved by examining only 4% of the entire database; a considerable reduction (25 times as compared to the linear constraint case) in the computational cost of retrieval. The evaluation results presented in this paper are still very preliminary. We plan to conduct more elaborate evaluation of our retrieval model against the other information retrieval models.

There are many potential applications of this work, the most direct one being to facilitate scalable video databases. It should be kept in mind that the retrieval model used here is not limited to indexing video data. Our retrieval model and the methods to maximize the expected size of the target set could have been implemented over *any* set of multimedia features. For instance, these features could have been derived from images or audio. An interesting implementation would be to use our principles on mixed media (for example, audio and video features) directly.

There are several system issues that have not been adequately dealt with in this work. The most important concern is the fact that we can only directly compute conditionals of the form $P(\alpha/\beta)$. The higher order conditionals (as explained earlier) could not be directly calculated due to prohibitive system costs. Instead we estimate conditionals of the form $P(\alpha,\beta/\gamma)$ using the lower order conditionals. We believe that some of these problems result due to using scalar quatization in this work. Using vector quantization would enable more accurate higher order estimates.

Another problem that we are currently working on is the multiple object query scenario. Our algorithm, optimizes the retrieval for each object in the query. To query for multiple objects, we simply perform a conjunction on the videos retrieved by each object query. This is clearly sub-optimal in the sense that it does not attempt to maximize the likelihood that all objects would be present in the returned result.

8. References

[1] S. Arya, D.M. Mount *Approximate Range Searching,* Proc. 11th Annual ACM Symposium on Computation Geometry, pp. 172-181, 1995.

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

[3] S. Berchtold, C. Bohm and H.P. Kriegel *The Pyramid-Technique: Towards Breaking the Curse of Dimensionality,* Int. Conf. on Management of Data, Seattle WA, Jun. 1998.

[4] T. Brinkhoff, H.P. Kriegel, R. Seeger *Comparisons of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Databases Systems,* Proc. 9th Int. Conf. on Data Engineering, pp. 40-49, Vienna Austria, 1993.

[5] S.F. Chang, W. Chen, J. Meng, H. Sundaram, D. Zhong *VideoQ: An Automated Content Based Video Search System Using Visual Cues,* ACM Multimedia '97, Seattle, Nov 8-14. 1997.

[6] W.S. Cooper *Some Inconsistencies and Misnomers in Probabilistic Information Retrieval,* SIGIR '91. Proc. ACM/SIGIR Conf. on Research and Development in Information Retrieval, pp. 57-61, 1991.

[7] R. Fagin *Combining Fuzzy Information from Multiple Sources,* Proc. ACM SIG-PODS '96 pp. 216-226, Montreal, Quebec, Canada, Jun. 1996.

[8] C. Faloutsos and K.-I Lin *FastMap: a Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets,* Proc. ACM-SIGMOD, pp. 163-174, San Jose, CA, May, 1995.

[9] M. Flickner et. al. *Query by Image and Video Content: The QBIC System,* IEEE Computer Magazine, Vol. 28, No. 9, pp. 23-32, Sep. 1995.

[10] J.H. Friedman, J.L. Bently, R.A. Finkel *An Algorithm for Finding Best Matches in Logarithmic Expected Time,* ACM Transactions on Mathematical Software, Vol. 3, No. 3, pp. 209-226, Sep. 1977.

[11] A. Guttman *R-trees: A Dynamic Index Structure for Spatial Indexing,* Proc. ACM SIGMOD, Int. Conf. On Management of Data, pp. 47-54, 1984.

[12] J. Hafner et. al. *Efficient Color Histogram Indexing for Quadratic Form Distance Functions,* IEEE Trans. On PAMI, Vol. 17, No. 7, pp. 729-736, Jul. 1995.

[13] J.M. Hellerstein *Optimization Techniques For Queries with Expensive Methods,* to appear, ACM Transactions on database systems.

[14] J.M. Klienberg *Two Algorithms for Nearest-Neighbor Search in High Dimensions,* Proc. 29th ACM Symposium on Theory of Computing, 1997.

[15] S. Miyamoto *Fuzzy Sets in Information Retrieval and Cluster Analysis,* Kluwer Academic Publishers, Boston MA, 1990.

[16] M. Ortega et. al. *Supporting Similarity Queries in MARS,* Proc. Of ACM Multimedia '97, pp. 403-413, Seattle, Washington , Nov. 8-14, 1997.

[17] A. Pentland, R. Picard and S. Scarloff *Photobook: Content-Based Manipulation of Image Databases,* IJCV, Vol. 8, No. 3, pp. 233-254, 1996.

[18] G. Salton, M.J. McGill *Introduction to Modern Information Retrieval* McGraw Hill Computer Science Series, Nov. 1983.

[19] T. Siedl H.P. Kriegl *Optimal Multi-Step k-Nearest Neighbor Search,* Proc. ACM SIGMOD Int. Conf. on Management of Data, Seattle WA, Jun. 1998.

[20] J. R. Smith, S.F. Chang, *VisualSEEk: A Fully Automated Content-Based Image Query System,* ACM Multimedia '96 Boston, MA, Nov. 1996.

[21] H. Sundaram S.F. Chang *Efficient Algorithms for Video Retrieval,* Tech Rep. Dept. of Electrical Engineering, Nov. 1998.

[22] H. Tamura, S. Mori *Textural Features Corresponding to Visual Perception,* IEEE Trans. on Systems, Man, and Cybernetics, Vol. 8, No. 6, Jun. 1978.

[23] A. Thomasian, V. Castelli, C.S. Li *CSVD: Approximate Similarity Searches in High Dimensional Spaces Using Clustering and Singular Value Decomposition,* Proc. Of the SPIE Conf. On Multimedia Storage and Archiving Systems III, SPIE Vol. 3527, pp. 144-154, Boston MA, Nov. 1998.

[24] D.W. White and R. Jain *Similarity Indexing: Algorithms and Performance*, Proc. Of the SPIE Conf. On Storage and Retrieval for Image and Video Databases IV, pp. 62-75, San Jose, CA, Feb. 1996.

Appendix

A Relating Weights to Metric Thresholds

In this section we show how the metric threshold for each feature is determined using the weight ω set by the user. The feature weight represents in the view of the user, the relevance of that feature with respect to the query. We interpret a large feature weight to imply that the feature is very discriminatory implying that we need only retrieve a small set. Hence, we use the following equation for the metric threshold:

$$\delta(\omega) = \frac{1}{a + b\,\omega},$$

where, a and b are constants such that $\delta(1) = d_o$ and $\delta(0) = 1.0$. Note that $\delta \in [d_0, 1.0]$. In our system, d_o has been empirically set at 0.2. Note that a metric threshold of 1.0 implies that we shall retrieve the entire database. This must be so since (ω=0) the feature has been deemed to be non-discriminatory.