Content-Based Image Indexing

Tzi-cker Chiueh
Computer Science Department
State University of New York at Stony Brook
Stony Brook, NY 11794-4400

chiueh@cs.sunysb.edu

Abstract

We formulate the content-based image indexing problem as a multi-dimensional nearest-neighbor search problem, and develop/implement an *optimistic vantage-point tree* algorithm that can dynamically adapt the indexed search process to the characteristics of given queries. Based on our performance study, the system typically only needs to touch less than 20 % of the index entries for well -behaved queries, i.e., when the query images are relatively close to their nearest neighbors in the database. We also report in this paper the results of extensive performance experiments, which characterize the impacts of various configuration and workload parameters on the performance of the proposed algorithm.

1 Introduction

Multimedia information systems integrate text, images, audio/video, and graphics data in a single framework. They also require computer systems designers to re-visit certain systems issues with completely different design tradeoffs than conventional ones. One example is to put image/video into a database system. Query facilities are typically regarded as the very basic service that a database system should offer. Specifically, users should be able to access the data in the database based either explicitly on the data entity's name or implicitly on the data's contents. The mechanisms for content-based access to alphanumeric data have been extensively studied and are now considered relatively well understood. However, the notion of content-based image/video access at this point still remains elusive at best.

There are two fundamental problems associated with content-based query systems for imaging data: how to specify queries, and how to access the intended data efficiently for given queries. For traditional database systems, the semantics of content-based access are finding data items that are exact matches of specified keywords in the queries. As a result, it is relatively straightforward to specify queries and to search for the matching data items. For image/video database systems, both query specification and data access become much harder. One potential approach to specify image queries is to use a text-based formalism to describe the image contents, based on the assumption that it is always possible to abstract high-level symbolic descriptions from low-level image data. Obviously this is not even possible based on natural languages, let alone any formal languages. For example, it is not clear how one can describe the texture or color of an image satisfactorily only based on textual descriptions. In some sense, this approach relies heavily on the advancement of knowledge representation. One possibility is to adopt the query-by-example paradigm [OZSO93]. That is, users can use either hand-drawn sketches or existing images as query templates, and ask the system for images similar to the query images. This is the so-called "like-this image retrieval." Although this might seem to solve the image description problem, new problems arise. For example, using existing images as queries poses the question of how one gets the first image. Admittedly, it is possible to use keyword-based queries or data browsing techniques to access the first image that is reasonably close to the desired ones. Whether these techniques can satisfy the need of most interesting applications remains to be seen. User-drawn sketches have similar problems of accuracy and ease of use, with an additional complication of cursive pattern recognition.

[0]**Proceedings of the 20th VLDB Conference Santiago, Chile, 1994**

Even if it is possible to specify query images precisely, there is still the issue of "like-this semantics." In other words, when users present a query image and ask for similar images, how exactly is similarity defined? Since an image can be characterized in various ways, it is almost impossible for the system to retrieve the desired images without further specifications. Consequently, even with image-based queries, the users still need to specify particular aspects of the query images, e.g., color and/or shape, on which the system should emphasize while performing similarity search.

Retrieving images similar to a given query image is a rather old problem in the field of pattern recognition. However, there are several important differences between image database retrieval and pattern recognition. First, the images in an image database may contain arbitrarily complex scenes and therefore are typically much more complicated than those in pattern recognition systems, which typically deal with artifacts or highly abstract figures. Second, the number of images in an image database is usually much larger than that of a pattern recognition system. In fact, most pattern recognition systems only need sequential searching simply because the number of candidate patterns to match is relatively small. Third, sometimes pattern recognition systems will make high-level interpretations directly based on features extracted from the given patterns. In these cases, it is imperative that the feature extraction procedure be absolutely correct. In contrast, the realistic expectation for an image database system's access mechanism should be to filter out un-related data, leaving further identification and semantic interpretation of image objects to human users. More precisely, pattern recognition aims at identifying positive matches, whereas an image database access method is considered successful when enough negative matches are ruled out to the extent that interactive browsing becomes feasible. As a corollary, the accuracy requirements for feature extraction is less stringent for image database systems. Actually, a feature extractor is considered useable for image database systems as long as the errors made by the feature extractor in query images and candidate images in the database are consistent.

The first and third differences will impact the image processing algorithms that underly content-based image access systems. They are beyond the scope of this paper and will not be discussed further. The second difference requires the image database system to have an indexing structure for faster access response. When designing the indexing mechanism, one should take advantage of the subtle distinction between positive identification, as in traditional database access problems, and negative exclusion, as in image data access. In addition, because image features are inherently multi-dimensional objects and because it is very rare to have exact image match, the image index structure must efficiently support multi-dimensional nearest-neighbor search.

The goals of this project are twofold. First, we aim at developing an efficient indexing mechanism for retrieving images solely based on the images' contents or features. We have designed and implemented an *optimistic vantage-point tree* index structure and examined its various performance characteristics. Second, we want to study the interaction between feature extraction (image processing) and feature indexing (data management). In particular, we'd like to deduce the preferred properties of those image features that facilitate the indexed search process, in order to guide the development of more sophisticated feature extraction algorithms. However, this paper will *not* address any image processing issues related to feature extraction and representation.

The remaining sections of the paper are organized as follows. Section 2 gives a formulation of the image index problem, and discusses the design issues involved. Section 3 describes a multi-dimensional nearest-neighbor search algorithm in a disk-based environment, and its various optimizations. Section 4 presents extensive performance results from a prototypical implementation of the algorithm. In Section 5, earlier works on image database indexing are reviewed to set the contributions of this work in perspective. We summarized major ideas and results in Section 6 with an outline of future work.

2 General Framework

An image is a two-dimensional array of pixels, each of which is represented with certain precision, e.g., using 8 bits or 24 bits per pixel. From raw images, there are various ways of building up high-level abstractions; for example, color distributions based on histograms, and object contours based on thresholding. This process is usually referred to as *feature extraction.* We call each scalar piece of such high-level abstractions a *feature*. So an example feature may be one of the component values of a color attribute, or a component coordinate of a shape descriptor. The set of features that comprise a coherent high-level interpretation form a *feature class.* Example feature classes are color, texture, and shape. From the standpoints of both data management and pattern recognition, it is better to organize database images on a feature-class by feature-class basis, rather than based on a lumped feature set that includes every image feature. There are two reasons to justify this proposition. First, combining multiple feature classes usually doesn't make too much sense semantically, and users actually lose the flexibility of accessing data based on individual feature class. For example, users may want to access the images only based on color information. An access mechanism based on aggregate feature sets tend to complicate this type of access. Second, suppose each feature is treated as a dimension, then combining multiple feature classes increases the number of dimensions. This will certainly aggravate the "dimensionality curse" problem associated with many multi-dimensional search algorithms, which says that as the dimensionality increases, the efficiency of the search algorithm decreases.

We assume that images in the database and query images go through the same feature extraction process. So every image is represented as a K-dimensional feature vector, where K is the number of features used to represent an image. The image search problem can then be formulated as follows. Let S denotes the set of feature vectors representing the images in the database. Let E^K designates the K-dimensional feature space, then $S \subseteq E^K$. Given a query image's feature vector q, find the feature vector $p \in S$ such that $d(p, q)$ is the minimum, where $d(.)$ is the distance metric used in the feature vector space. The goal of image indexing is to sort the feature vectors in S so that servicing a query doesn't always need a sequential scan through the entire database.

It is clear that the choice of E^K and $d(.)$ could significantly affect the performance of the search mechanism. For example, it is possible that the data image that is closest to the query image in the feature space according to $d(.)$ is *not* necessarily the most similar data image according to visual perception or other *objective* criteria. Knowing that image similarity criterion is itself an active research problem, in this paper we will assume that the accumulated sum of pixel-by-pixel differences is the ultimate similarity metric between two images. In this model, E^K, $d(.)$, and the similarity metric are all domain-dependent, and should be tailored to individual applications. However, the design of the index mechanism should be generic enough to accommodate different choices of these parameters.

The above framework in itself doesn't make any assumption about the capabilities of the image processing subsystems that extract features. Therefore it is not restricted to whole-image matching. As long as there are ways to segment objects out of images, it can be equally applicable to subimage or object matching. Similarly, if the features chosen have invariance properties, the index mechanism can also support invariance across scaling, translation, or rotation transformations. In summary, the indexing mechanism is completely independent of whether the image features allow sub-image matching or transformation invariance.

3 Indexing for Multidimensional Nearest-Neighbor Search

Given a point in the K-dimensional space, searching for its nearest neighbor can be facilitated by sorting points in the database first. As mentioned earlier, the design goal of an image indexing scheme should focus on eliminating unlikely candidates rather than pin-pointing the targets directly. So an important requirement for image indexing schemes is that a data point not be the nearest neighbor when the index subsystem declares that it is not. That is, the indexing scheme must work conservatively.

Conventional spatial index structures use some kind of partitioning methods to divide the multidimensional vector space into partitions. The goal is to ensure that each partition have approximately the same number of data points so that finding the nearest neighbor of a given query point only needs to touch a small number of partitions. These partitioning methods are almost always based on *absolute coordinate values* of the vector space. For example, a partition in a K-dimensional hypercube is characterized by K pairs of coordinate values, each of which specifies the covering interval in the respective dimension. This type of partitioning structure is useful for queries based on absolute coordinates, such as range queries, e.g., find all the 2D points inside the rectangle defined by [X1, X2] and [Y1, Y2]. However, it is not so useful for nearest-neighbor search because the search structure in general doesn't maintain the distance information between points within a partition and the partition's boundaries. As it turns out, this information is critical in pruning the search space for multi-dimensional nearest-neighbor search.

Intuitively, since nearest-neighbor search by definition is to look for the one point with the minimum point-to-point distance, it is only natural to conceive partitioning methods that are based on *relative distance* rather than absolute coordinate values. This is exactly the central idea behind the *vantage-point (VP) tree* method proposed by P. Yianilos [YIAN92] and will be explained in the following subsections.

3.1 Initial Construction

In conventional multi-dimensional data structures such as *K-d trees*, the data set is first projected along each dimension, and the median of the projection values along the dimension that has the maximum spread is chosen as a cut point. The data set is partitioned into two subsets by comparing each data point's projected value in the corresponding dimension with the

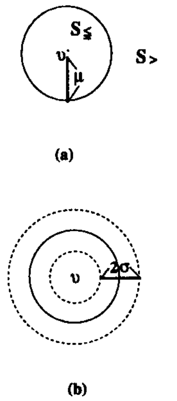

Figure 1: Illustration of the Vantage Point

cut point. The VP tree method abandons the projection approach and bases the partitioning on the relative distances between the data points and a particular *vantage point*.

Let's start with the binary partitioning case, and then generalize to n-ary partitioning for disk-based environments later. Assume a particular data point in the data set, v, is chosen as the vantage point. Then the system computes the distance between any other point in the data set and v, and gets $d(p,\ v)$, for $p \in S - \{v\}$. Find the median μ among the $d(p,\ v)$'s and partition the data set into two halves according to μ. Although it may seem that the VP tree method just uses a different partitioning metric, the fact that it takes a relative-distance approach significantly improves the pruning effectiveness.

To illustrate this concept, considers a 2D vector space as shown in Figure 1(a), where v is the vantage point and μ is the median of the distance values of all the other points with respect to v. Therefore the data set is partitioned into two subsets: $S_\leq$ and $S_>$, corresponding to the data points whose distance to v is smaller than or equal to μ, and larger than μ, respectively. Suppose users request the nearest neighbor of a query point q with the requirement that the maximum distance between a query point and its nearest neighbor be smaller than a specific threshold, σ. That is, if it turns out that the distance between a query point and its nearest neighbor is greater than σ, the nearest neighbor is meaningless and therefore not interesting. With these requirements, it can be shown that to locate the nearest neighbor of q, the system only needs to explore both $S_\leq$ and $S_>$ if and only if q is enclosed within the two concentric circles with radii of $max[0,\ \mu - \sigma]$ and $\mu + \sigma$ respectively, as shown by the dashed circles in Figure 1(b), i.e., when $\mu - \sigma < d(q, v) \leq \mu + \sigma$. Otherwise, the system only needs to explore one of them, thus effectively pruning one half of the search space. This pruning of the search space is based on the principle of triangular inequality. Specifically, if $d(q,\ v) \leq \mu - \sigma$, for $p \in S_>$, the distance between p and q is lower-bounded by

$$\begin{aligned} d(p,\ q) &\geq ||d(p,\ v)| - |d(q,\ v)|| \\ &\geq ||d(p,\ v)| - (\mu - \sigma)| \\ &> |\mu - \mu + \sigma| \\ &= \sigma \end{aligned} \qquad (1)$$

Therefore $S_>$ can be ignored when $d(q,\ v) \leq (\mu - \sigma)$. Similarly, if $d(q,\ v) > \mu + \sigma$, for $p \in S_\leq$, the distance between p and q is lower-bounded by

$$\begin{aligned} d(p,\ q) &\geq ||d(q,\ v)| - |d(p,\ v)|| \\ &\geq ||d(q,\ v)| - \mu| \\ &> |\mu + \sigma - \mu| \\ &= \sigma \end{aligned} \qquad (2)$$

Therefore $S_\leq$ can be pruned away when $d(q,\ v) > \mu + \sigma$. When $S_\leq$ and $S_>$ are approximate in size, each such pruning effectively cuts the search space in half.

This partitioning method is applied to each resulting partition recursively until the number of data points in a partition is small enough that the overhead of sequential scans becomes manageable. Therefore, the entire data set is also organized as a tree as in other spatial data structures. The difference is that at each level, a distinct vantage point is chosen to map the other data points in the subset, rather than simple projections based on absolute coordinate values.

To search for the nearest neighbor of a query point, the system first computes the distance between the query point and the vantage point associated with the root of the tree, and determines which subset(s) to explore next. This process proceeds recursively until the nearest neighbor is found or no such nearest neighbor whose distance to the query point is smaller than σ is reported. The overall pruning effect is multiplied at each level as the system traverses down into the VP tree. An optimization to the basic traversal algorithm is that during the traversal, if the distance between the query point and the current nearest neighbor candidate is smaller than σ, then σ is reduced to that distance value. This way subsequent search steps can avoid unnecessary probing.

It is clear that the choice of vantage points at each level of the VP tree and the value of σ play an important role in the performance of the indexing algorithm. An ideal vantage point should exhibit the following characteristic: The distribution of the distance values between other data points and the vantage point is close to a uniform distribution. Intuitively, this minimizes the number of data points in the concentric regions, and therefore reduces the probability of having to explore both subtrees. Given a data set, finding the optimal vantage point is exceedingly expensive computationally. Therefore, one uses a randomized algorithm to approximate the ideal vantage point. It has been shown in [YIAN92] that this algorithm works reasonably well in practice. The basic algorithm is as follows:

```
Pick a set of candidate vantage points
from the data set;
For each candidate vantage point
   Pick a subset of sample points from the
   data set;
   Compute the distance values from the
   vantage point to each sample point;
   Calculate the mean and the standard
   deviation of these distance values;
Endfor
Choose the candidate vantage point with
the maximum standard deviation;
```

The choice of σ represents the tradeoff between the likelihood of locating the nearest neighbors and the searching efforts. When σ is small, the concentric region becomes smaller, and therefore the probability of having to explore both subtrees is reduced. However, smaller σ also means that the maximum allowable distance between a query point and its nearest neighbor is reduced, making it more likely to reject the query point's nearest neighbor. Ideally, a default value of σ should be set by the system after analyzing the the distance value distributions. It is also desirable to decrease σ during the search process by replacing it with the distance between the current nearest neighbor candidate and the query point if that distance is smaller than initial σ. This last point argues for a depth-first, as opposed to breadth-first, order of searching through the VP tree because the former tends to reach the leaves of the tree faster and thus is more likely to reduce σ.

It is essential to pay attention to the characteristics of I/O devices for an index subsystem to be useful in a database environment. Because magnetic disk-based systems typically fetch a large chunk of data in each access to amortize the long fixed overhead, the indexing scheme must be designed to exploit I/O devices' capabilities. A universal technique as used in B-tree algorithms is to increase the branching factor of the tree. This not only improves disk access efficiency by packing sibling nodes into a disk page, but also significantly improves the pruning effect since it is now possible to prune $\frac{Branch-1}{Branch}$ of the data set, where *Branch* is the branching factor and is usually much larger than 2.

The N-ary VP tree construction algorithm is similar to the binary tree case. For a given data set, the distance values between the chosen vantage point and

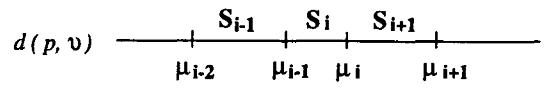

Figure 2: Partitions According to Distance Values to the Vantage Point

other data points are first computed. The data set is then split into N subsets, S_i, i = 1 to N, according to the distance values, preferably making each of S_i's roughly the same size. Let's use μ_i, i = 1 to N-1, to denote the boundary distance value between S_i and S_{i+1}. As shown in Figure 2, $\mu_{i-1} < d(p, v) \leq \mu_i$, for all $p \in S_i$, assuming that $\mu_0 = 0$. Given a query point p, the system needs to explore S_i as long as

$$\mu_i - \sigma < d(q, v) \leq \mu_{i+1} + \sigma \qquad (3)$$

for i = 1 to N. These conditions are also based on the triangular inequality principle and can be derived in a way similar to the binary case. So far the discussion focuses on the initial tree construction. A database system must also support dynamic insertion and deletion of data items. Therefore an incremental modification mechanism of the VP tree is needed to reflect the run-time changes in the database back to the index data structures. These are discussed in the next subsection.

Due to space constraints, the descriptions of incremental modification to the VP tree due to insertion and deletion are omitted and can be found in [CHIU94].

3.2 Optimization

To address the "dimensionality curse" problem, the index subsystem must use as few dimensions as possible. However, mapping an inherently high-dimension data set into a low-dimension space tends to lose the information that distinguishes the data items. Consequently the indices based on low-dimension feature sets may become less discriminative than those based on high-dimension feature sets. In particular, the nearest neighbors found through low-dimension indices are less likely the *true* nearest neighbors than those found through high-dimension indices. Theoretically, it should be possible to select a set of features with the maximum distinguishing power from an image processing standpoint. In practice, however, this is still considered rather far-fetched even for research systems.

One potential solution is to use the *divide and conquer* approach. The idea is to to use multiple index trees, each of which is based on a different subset of features, e.g., color, shape, or texture. With this structure, each index tree is based on a relatively small number of features, and therefore can avoid the "dimensionality curse" problem. On the other hand, the fact that multiple index trees collectively employ a large number of features should improve the overall distinguishing capability of the index system. Multiple index trees also make it easier to identify the distance metric function *d(.)* for each feature subset because the features in the subsets are presumably more correlated with one another than with those in other feature subsets. Moreover, it is easier for the system to handle queries that emphasize particular feature classes, e.g., image queries based on colors only.

The key to the success of multi-tree index search is that there exists a way to combine the search results from each index tree. Consider the following simple example. Suppose we have a five-dimensional feature space, supported by two index trees based on the first two and the last three features, respectively. Given a query point (0, 0, 0, 0, 0), the nearest neighbors in the corresponding feature subspaces are (0, 0, 100, 100, 100) and (100, 100, 0, 0, 0) respectively. However, the true nearest neighbor is actually (10, 10, 10, 10, 10). This example illustrates that the result of a global nearest neighbor search is *not* a simple AND or OR of the nearest neighbors found from the component index trees. One solution is to use a larger σ value for the searches in each component index tree to find all the neighbors that are within σ distance away. The hope is that the true nearest neighbor belongs to each of the result sets obtained from the searches of the component indices, and therefore could be identified by finding the intersection of these result sets.

The other important issue not addressed in [YIAN92] is the choice of σ. Presumably, values of σ should be completely transparent to users since the underlying feature set and the distance metrics are typically hidden from the end users. Therefore, the first question is how the default value of σ is chosen given a feature database. [YIAN92] implicitly assumes that users are responsible for the choosing. However, we believe that this is unrealistic because the index structures, including E^K, $d(.)$, and the distribution of S are supposed to be completely transparent to the users. It's not clear that users are able to make *any* decision of the initial σ value without these information, let alone the optimal σ. We have developed an adaptive algorithm to automate the choice of σ and achieve reasonable performance.

Ideally, there should be a different σ value associated with each vantage point within the index tree, and the value should be determined based on the distribution of the distance values with respect to that vantage point. Intuitively, the default σ value is chosen in such a way that unnecessary probing is minimized. To reduce search efforts, one makes the optimistic assumption that *a query point's nearest neighbor is always very close to the query point.* This assumption translates into using the smallest possible σ values. With respect to a vantage point v, assume that each partition S_i is characterized by the lower and upper distance value bounds $LO[i]$ and $HI[i]$. During the traversal, it is not necessary to explore S_i as long as $d(q,\ v) > HI[i] + \sigma$ or $d(q,\ v) < LO[i] - \sigma$. Then the default $\sigma(v)$ value for a vantage point v is chosen to be

$$\sigma(v) = \max_{i=1}^{N-1} \frac{LO[i+1] - HI[i]}{2} \tag{4}$$

Such a choice of σ guarantees that any $d(q,\ v)$ value will fall within at least one of $[LO[i] - \sigma,\ HI[i] + \sigma]$, and therefore the traversal could proceed with at least one partition. In our implementation, we use the same σ for every vantage point in the tree, and the default σ value is chosen to be the maximum of all $\sigma(v)$'s determined by Equation (4).

The optimistic approach is based on the observation that there is typically a big difference between the time needed to locate the nearest neighbor and the time needed to verify that it is indeed the true nearest neighbor. Using smaller initial σ values significantly reduces the "verification" work when the query point is indeed close to its nearest neighbor. On the other hand, because the initial σ value is chosen optimistically, there must be a fall-back mechanism to handle those cases in which the assumption is not valid, i.e., when a query point's nearest neighbor is not very close to the query point. One can either use an additive or multiplicative algorithm to adjust the values of σ. In other words, if the nearest neighbor search fails for a given σ, then σ is modified according to

$$\sigma_N = \sigma_{N-1} + \alpha \tag{5}$$

or

$$\sigma_N = \sigma_0 * \gamma^{N-1} \tag{6}$$

where α and γ are additive and multiplicative constants, and N is the number of trials that have failed. The rationale of this formula is to minimize the average overall search efforts by balancing between the search efforts of individual trials and the number of trials for each query point. Our implementation made the following optimization that further improves the performance: When a leaf node is visited, the point in the node that is closest to the current query point is located. Therefore, when a leaf node is visited again during subsequent trials for the same query point, the system only needs to examine the node's associated nearest point without further touching any point in that leaf node, essentially re-using the efforts performed in previous trials. This technique reinforces the power of the proposed optimistic approach.

4 Performance Results

4.1 Experiment Set-up

We take a sequence of seven 240x352 video frames, and decompose each frame into 1,320 8x8 blocks. From each block, four column average values, four row average values, and an overall average value (i.e., the DC value of a block) are extracted to form a nine-element feature vector. The distance metric we choose is the infinity norm, i.e., sum of absolute differences. The blocks in the first five frames form the database against which queries are run. Therefore there are totally 6,600 feature vectors in the database. We use three sets of queries. The first set consists of the blocks from the last two frames of the original video sequence. Because of the nature of video sequences, each of these blocks is considered *close* to its nearest neighbor in the database. The average distance between a query block in this set and its nearest neighbor is 16.39. The second set of query blocks are derived by injecting *small* uniformly distributed random noises to each block in the first query set. This set is called the *median* set and the average distance between a query block in this set and its nearest neighbor is 32.64. The third set of query blocks are derived by injecting *large* uniformly distributed random noises to each block in the first query set. This set is called the *far* set and the average distance between a query block in this set and its nearest neighbor is now 217.83. For each set, there are 2,640 query blocks, and the final performance numbers are average values from these 2,640 runs.

Three performance metrics are adopted. They are the percentage of index entries (PIE) that are accessed to locate the nearest neighbor, the number of trials (NT) per query, and the actual running time (ART). PIE measures the average portion of the index tree that needs to be examined to service a query. NT gives how many times the index subsystem needs to enlarge the σ value before locating the nearest neighbor. ART calibrates the total computation cost for each query. Because the database is loaded into main memory entirely, our experiment didn't reflect accurately the disk I/O cost. However, the percentage of index entries accessed metric should reflect the I/O overhead to a certain extent. On the other hand, the processing time measurement is fairly accurate because we use a dedicated machine for these experiments, thus minimizing deviations due to operating systems or multiprogramming. In order to understand the impact of the dimensionality of feature vectors, we also construct a

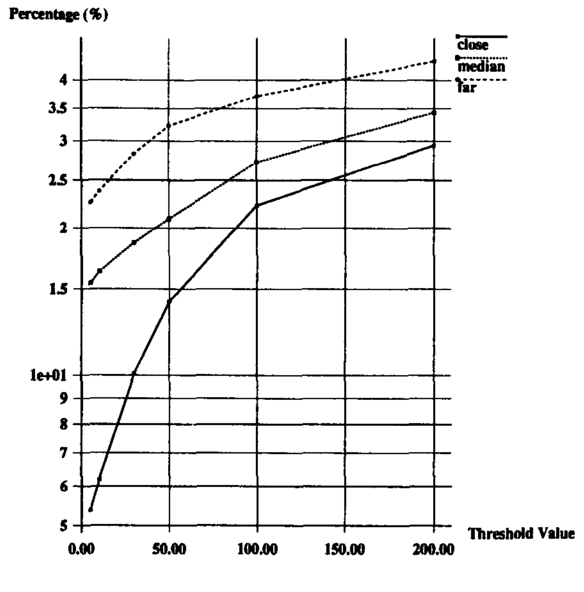

Figure 3: *Percentage of Index Entries Accessed vs. Initial* σ

17-element feature vector database. In this case, all of the eight column and eight row average values from a 8x8 block are incorporated into the feature vector.

4.2 Results and Analysis

The first set of results represent the baseline case, which corresponds to the nine-feature-vector workload with the additive σ value adjustment method. Figure 3, 4, and 5 show the percentage of index entries accessed, the number of trials per query, and the actual running time versus the initial σ value under this workload. The X-axis represents different initial values of σ, and the Y-axis is on a log scale for the PIE metric. Although the number of iterations per query is decreasing with larger σ (Figure 4), the percentage of index entries accessed (Figure 3) actually increases. This demonstrates the effectiveness of the dynamic σ adjustment method, compared to schemes based on fixed σ. The actual running time (Figure 5) exhibits an interesting tradeoff between the processing overhead associated with multiple iterations and the access overhead related to touching the index entries. The graph clearly shows an optimal design point for each workload, which seems to correlate very well with the average distance between the vectors in the respective query set and their nearest neighbors. For example, the optimal point for the *close* query set occurs when the initial σ value is between 10 and 30, and the average distance of the *close* query set is 16.39.

The second set of results compare the performance difference between the multiplicative and additive methods for adjusting σ. For our experiment, we

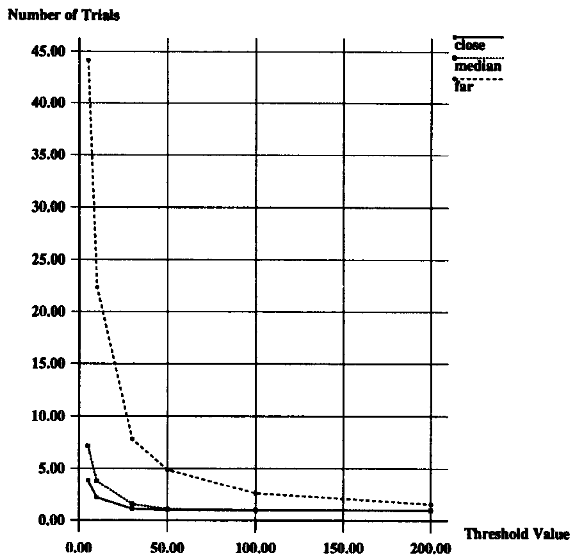

Figure 4: *Number of Trials per Query vs. Initial σ*

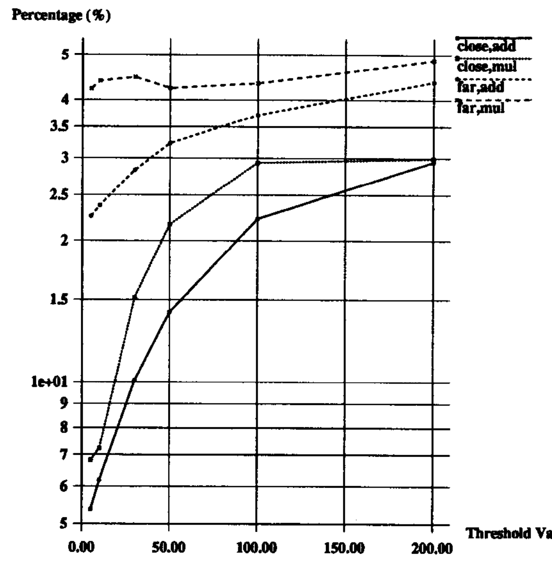

Figure 6: *Percentage of Index Entries Accessed vs. Initial σ. Comparison between Multiplicative and Additive σ adjustment methods, $\alpha = \sigma_0, \gamma = 2$*

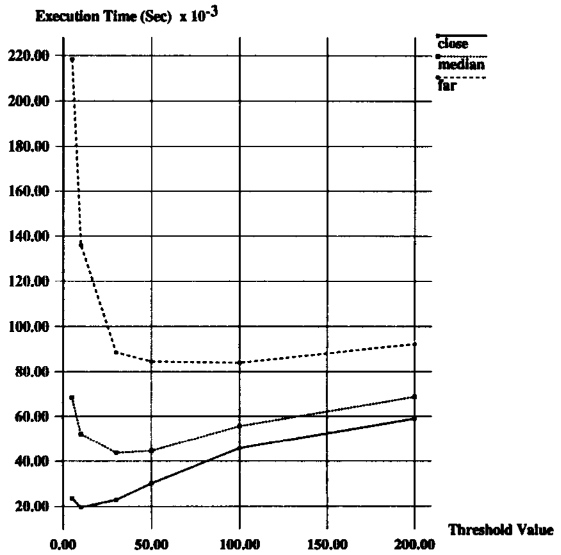

Figure 5: *Execution Time per Query vs. Initial σ*

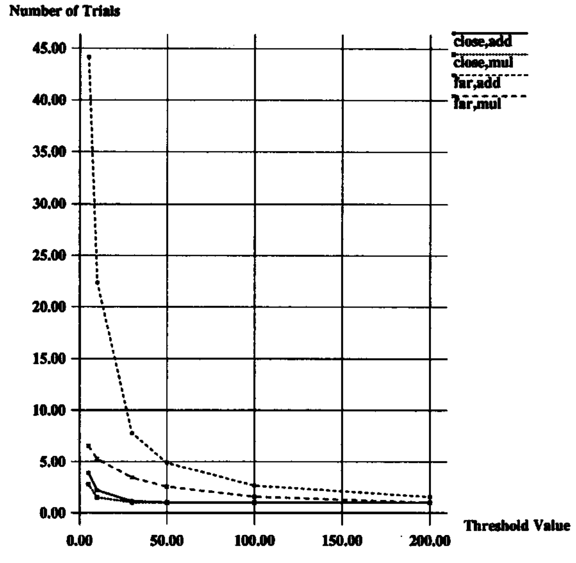

Figure 7: *Number of Trials per Query vs. Initial σ. Comparison between Multiplicative and Additive σ adjustment methods, $\alpha = \sigma_0, \gamma = 2$*

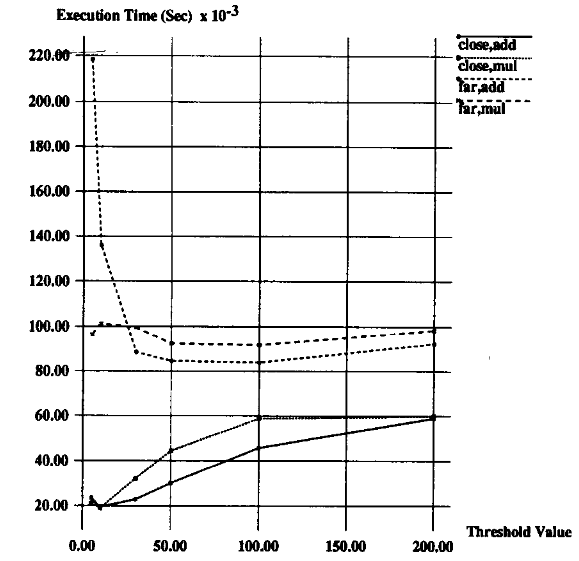

Figure 8: *Execution Time per Query vs. Initial σ. Comparison between Multiplicative and Additive σ adjustment methods, $\alpha = \sigma_0, \gamma = 2$*

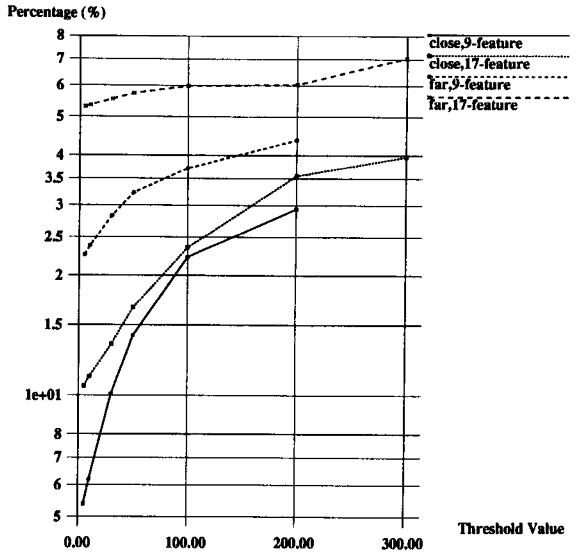

Figure 9: *Percentage of Index Entries Accessed per Query vs. Initial σ. Impact of the dimensionality of the feature space*

choose the multiplicative constant γ to be 2 and the additive constant α to be the initial σ value. So it may well be the case that the following comparison is valid only for this particular choice, although the general trend of the performance curves should remain largely unaffected. Generally speaking, although the multiplicative method requires fewer iterations than the additive method (Figure 7), the former actually touches more data (Figure 6) and take more time (Figure 8) than the latter. The only exception is when the queries are drawn from the *far* set and the initial σ is small, where the multiplicative scheme performs significantly better (Figure 8). Because the multiplicative method represents a more responsive feedback mechanism than the additive one, the former should be more effective in cases where the distance between the query point and its nearest neighbor is large. It may be interesting to experiment with other choices of multiplicative and additive constants and compare their performance behavior.

The next set of results show the performance impacts of the dimensionality of the feature space on the performance of the indexing system. Here we assume the σ adjustment method is additive, but the multiplicative case shows similar results. In the case of the *close* query set, the PIE (Figure 9) and the NT (Figure 10) are relatively close to each other for the 9-feature case and 17-feature case. But the ART (Figure 11) shows at least a factor of two difference. In the case of the *far* query set, the differences between the *far* and *close* sets in all three metrics are more pronounced in

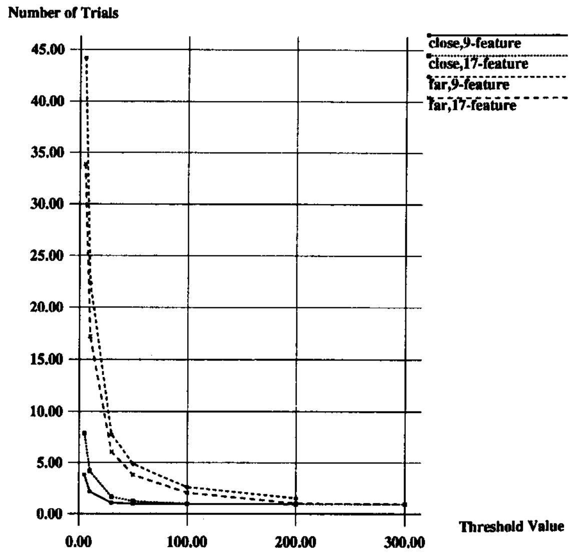

Figure 10: *Number of Trials per Query vs. Initial σ. Impact of the dimensionality of the feature space*

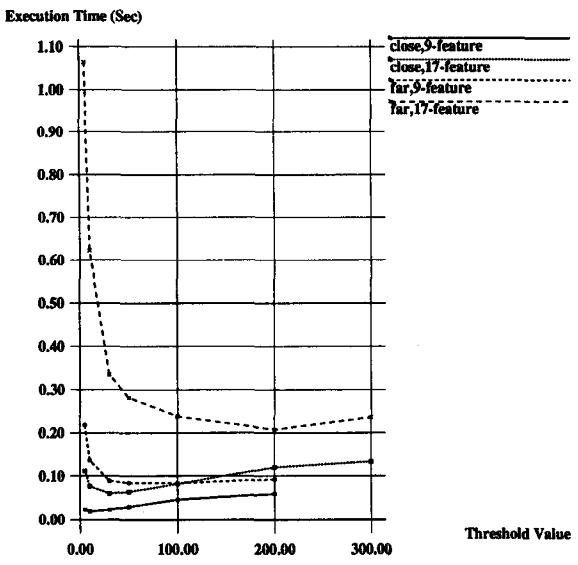

Figure 11: *Execution Time per Query vs. Initial σ. Impact of the dimensionality of the feature space*

the 9-feature case than in the 17-feature case. This result implies that the dimensionality curse problem is even more serious when presented with the *far* queries than *close* ones.

The last experiment explores the performance impact of the branching factor of the index tree, or alternatively, the number of index entries in a page. Here the X-axis represents the branching factor and the curves represent different combinations of workload parameters, σ adjustment methods, and initial σ values. In the case of PIE, Figure 12 (the additive case) and Figure 13 (the multiplicative case) display remarkable similarity in terms of the general trend. Although larger branching factors imply more effective pruning, this is only the case when the database is relatively large. Because the test database is relatively small, larger branching factors could actually need to access more index entries at the intermediate levels of the tree because the coverage of each partition is so small that there are actually more than one μ_i's can satisfy Equation (3). This explains why in Figure 12 and Figure 13, the percentage of index entries increases with the branching factor and eventually levels off. The other performance problem associated with larger branching factors is that it requires more comparisons at each level of the tree to determine which partitions need further exploration. This effect shows up in the average execution time (Figure 14 and Figure 15). The curves go up monotonically with increasing branching factors. This means that the larger CPU processing overhead associated with larger branching factors outweighs the benefits of touching less data. Of

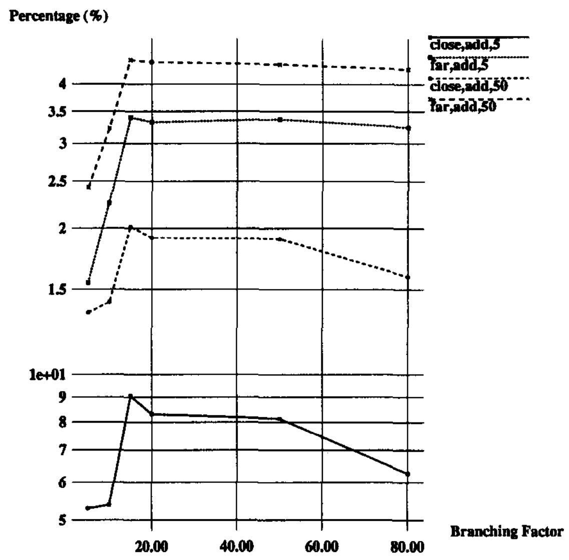

Figure 12: *Percentage of Index Entries Accessed vs. Branching Factor. Impact of the branching factor of the VP tree, with additive σ adjustment*

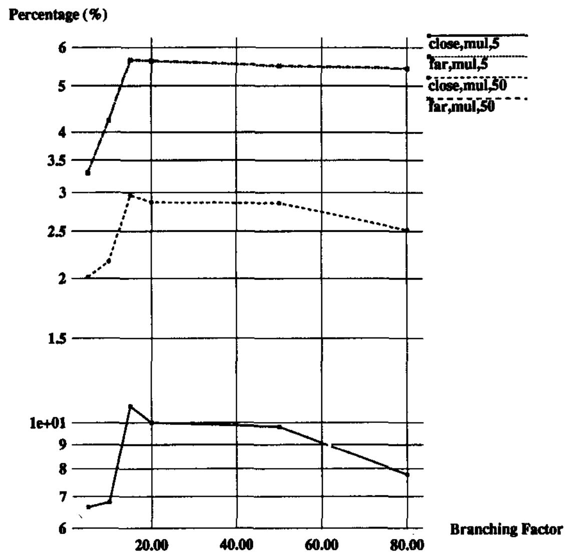

Figure 13: *Percentage of Index Entries Accessed vs. Branching Factor. Impact of the branching factor of the VP tree, with multiplicative σ adjustment*

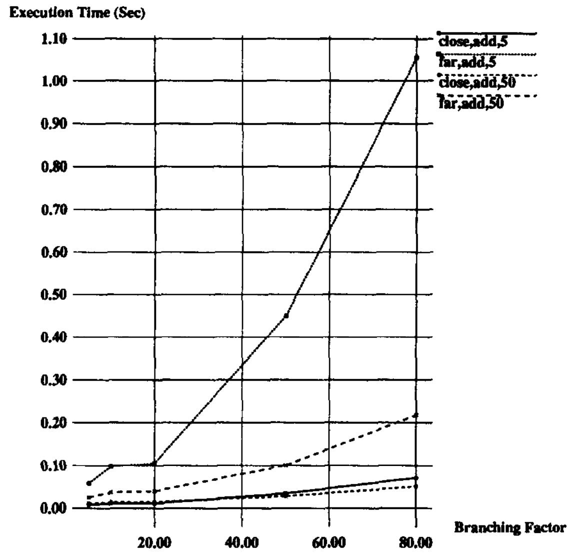

Figure 14: *Execution Time per Query vs. Branching Factor. Impact of the branching factor of the VP tree, with additive σ adjustment*

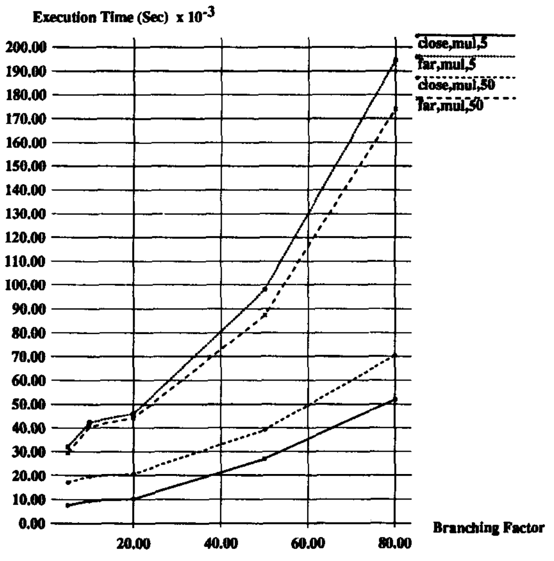

Figure 15: *Execution Time per Query vs. Branching Factor. Impact of the branching factor of the VP tree, with multiplicative σ adjustment*

course, since our experiment is completely based on main memory implementations, this conclusion may not hold when the data volume is so big that the I/O cost becomes dominant.

5 Related Works

The Vantage-Point Tree method is proposed in [YIAN92] mainly as a main-memory-based method for multi-dimensional nearest-neighbor search. No attempt has been made in that work to take advantage of the characteristics of image data and to apply it to a disk-based environment. The major contribution of our work is the development of the optimistic σ value adjustment mechanism that at once solves the problem of choosing the initial σ value and achieves the critical balance between recall and precision rates by dynamically tailoring the range of search to the characteristics of the queries. [ARYA92] proposes another algorithm to the multi-dimensional nearest-neighbor search problem based on a randomization approach. However, this algorithm doesn't seem to be as effective as the Vantage-Point Tree method. Both [JAGA91] and [MEHR93] describe experiments to retrieve image objects based on their shapes. However, the emphasis of these works is on the image representation schemes to support occlusion or partial matching, rather than on efficient indexing mechanisms to speed up the access. IBM Almaden's QBIC project [NIBL93] supports various mechanisms to do semi-automatic segmentation and interactive retrieval. However, relatively little emphasis has been put in efficient indexing for image objects. [GROS92][GROS89] described an indexing scheme very similar to the one presented here in that they also have a pruning mechanism based on the principle of triangular inequality. However, their index structure is still based on absolute feature values rather than the relative distance among feature vectors. As a result, the traversal through the index tree becomes unnecessarily complicated. Moreover, their method doesn't include a mechanism corresponding to our dynamic σ adjustment scheme. Consequently, their method probably won't perform as well as ours, especially when the distances between the query images and their nearest neighbors exhibit a large dynamic range.

6 Conclusion

Indexing improves the speed of image data access because it substitutes index manipulation for raw image manipulation, and because it reduces the amount of search efforts at run time. The first reason is probably more significant since the I/O and computation requirements of manipulating images could easily overwhelm most current workstation-class machines. We formulate the content-based image indexing problem as a multi-dimensional nearest-neighbor search problem, and choose a newly developed index algorithm called the *Vantage-Point Tree* method to solve this problem. This algorithm is particularly effective for nearest-neighbor search because it uses *relative distance* as the decomposition criterion rather than the absolute feature values used in conventional multidimensional spatial data structure. One of the most important performance parameters for this indexing algorithm is the choice of σ. We develop an optimistic algorithm to dynamically adjust the σ value in order to achieve a better balance between precision and recall rates against a wide variety of workloads. This algorithm also successfully relieves users of the burden of choosing σ values, an important factor that determines whether the indexing scheme is useable in practice. Based on our preliminary performance study, one only needs to touch less than twenty percent of the database for well-behaved queries, i.e., the query images are relatively close to their nearest neighbors in the database. We also perform extensive performance studies to investigate the impacts of various configuration and workload parameters on the performance of this algorithm.

As for future work, there are three possible directions that we are currently working on. First, exploring the interaction between feature selection and index structures. Right now we developed the image indexing scheme without regards to the nature of extracted features. In practice, the index algorithm can achieve the optimal performance only when the feature vectors assume certain characteristics. For example, in our case, the VP tree method works best when the feature vectors are uniformly distributed in the K-dimensional space. Especially in image databases, choosing the optimal feature set seems to us the single most important issue for solving the content-based image retrieval problem. Second, we plan to further explore the idea of multiple index structures and experiment with concurrent execution of multiple indexed searches. Lastly, we are interested in extending the current implementation to a disk-based environment and integrate with parallel I/O capabilities provided by advanced disk array technologies.

7 Acknowledgement

This work was performed while the author visited Siemens Corporate Research during the Summer of 1993. The discussions with the Image Database group led by Dr. Arding Hsu are greatly acknowledged.

8 Reference

[ARYA92] S. Arya, D. Mount, "Approximate Nearest Neighbor Queries in Fixed Dimensions," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 271-280, Orlando, Fla., 1992.

[CHIU94] T. Chiueh, "Content-based Image Indexing," ECSL TR-7, Computer Science Department, SUNY at Stony Brook, January 1994.

[GROS92] Grosky, W.I., et al. "A pictorial index mechanism for model-based matching." DATA and KNOWLEDGE ENGINEERING (1992) vol.8, no.4, p. 309-27.

[GROS89] Grosky, W.I., et al. "A pictorial index mechanism for model-based matching." PROCEEDINGS FIFTH INTERNATIONAL CONFERENCE ON DATA ENGINEERING, Los Angeles, CA, USA, 6-10 Feb. 1989.

[JAGA91] Jagadish, H.V. "A retrieval technique for similar shapes." 1991 ACM SIGMOD International Conference on Management of Data, Denver, CO, USA, 29-31 May 1991. SIGMOD RECORD (June 1991) vol.20, no.2, p. 208-17.

[MEHR93] R. Mehrotra, J. Gary, "Feature-Based Retrieval of Similar Shapes," International Conference on Data Engineering, '93, pp. 108-115.

[NIBL93] Niblack, W., et al. "The QBIC project: querying images by content using color, texture, and shape." Storage and Retrieval for Image and Video Databases. Held: San Jose, CA, USA, 23 Feb. 1993. PROCEEDINGS OF THE SPIE - THE INTERNATIONAL SOCIETY FOR OPTICAL ENGINEERING (1993) vol.1908, p. 173-87.

[OZSO93] Ozsoyoglu, G., et al. "Example-based graphical database query languages." COMPUTER (May 1993) vol.26, no.5, p. 25-38.

[YIAN92] P. Yianilos, "Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces," Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 311-321, Orlando, Fla., 1992.