VQ-Index: An Index Structure for Similarity Searching in Multimedia Databases [*]

Ertem Tuncel
Electrical and Computer Engineering
University of California, Santa Barbara
ertem@ece.ucsb.edu

Hakan Ferhatosmanoglu
Computer and Information Science
Ohio State University
hakan@cis.ohio-state.edu

Kenneth Rose
Electrical and Computer Engineering
University of California, Santa Barbara
rose@ece.ucsb.edu

ABSTRACT

In this paper, we introduce a novel indexing technique based on efficient compression of the feature space for approximate similarity searching in large multimedia databases. Its main novelty is that state-of-the-art tools from the discipline of data compression are adopted to optimize the complexity-performance tradeoff in large data sets. The design procedure optimizes the query access time by jointly accounting for both database distribution and query statistics. We achieve efficient compression by using appropriate vector quantization (VQ) techniques, namely, multi-stage VQ and split-VQ, which are especially suited for limited memory applications. We partition the data set using the accumulated query history, and each partition of data points is separately compressed using a vector quantizer tailored to its distribution. The employed VQ techniques inherently provide a spectrum of points to choose from on the time/accuracy plane. This property is especially crucial for large multimedia databases where I/O time is a bottleneck, because it offers the flexibility to trade time for better accuracy. Our experiments demonstrate speedups of 20 to 35 over a VA-file technique that has been adapted for approximate nearest neighbor searching.

Keywords

Approximate similarity searching, clustering, indexing, retrieved information reduction, retrieved set reduction, vector quantization.

1. INTRODUCTION

The term "similarity search" refers to seeking data objects in a database, which are most similar to a given query object. This problem is central in a wide range of applications in multimedia databases, which may contain images, video, text, music, etc. [43]. Another application is duplicate entry detection, e.g., elimination of almost identical variants of a document in the same database. The degree of similarity between two objects is often measured by a distance function, e.g., the Euclidean distance, operating on *feature vectors* extracted from the data. The user submits a query object to a search engine, and may either provide a distance threshold ϵ, or the number of objects k to be returned. These types of queries are called ϵ-range query, and k-nearest-neighbor (k-NN) query, respectively.

The feature vector dimensionality is usually very high and the search procedure is subject to Bellman's notorious "curse of dimensionality" [5], i.e., the search space grows exponentially with the number of dimensions. The complexity is further exacerbated by the fact that the number of entries in the database is very large. However, significant savings in disk I/O costs are possible if one allows for *approximate* search results. Usually, the extraction of feature vectors from the data objects is itself a heuristic process that attempts to approximately capture relevant information. Moreover, even if the feature vectors represent the original data with 100% accuracy, users would still differ in their perceptional capacity and needs, and hence in their similarity expectations. Thus, rather than incur the extremely high cost of an exact result, it is more cost-effective to develop a fast search engine that effectively outputs an approximate set.

An efficient approximate similarity search engine must reduce not only the number of feature vectors retrieved, but also the information retrieved about each feature vector [19]. In this paper, we employ this paradigm and develop a novel approximate k-NN search approach. To reduce the size of the retrieved set of vectors, we exploit the available accumulated query history. We cluster the query distribution and, for each cluster, collect into a file the union of data points most likely to be in the answer set. Hence, when a query is to be processed, an accurate answer can be found by retrieving only the data file corresponding to the cluster into which the query point falls. To reduce the information retrieved about each vector, on the other hand, the feature vectors in each file are quantized separately and efficiently. The idea of using quantization to overcome the curse of dimensionality was first proposed by Weber et al. in [46]. Although they used a very simple scalar quantization scheme, their method achieved superior performance for high dimensional data over other indexing techniques. In general, an efficient quantization scheme must achieve a good balance between quality of the vector reconstruction and the number of bits used to describe the vectors. The best quantization scheme in this respect is known as vector quantization (VQ) [20]. VQ has been used in several disciplines extensively, includ-

[*]This work is supported in part by the National Science Foundation under grants no. EIA-9986057 and EIA-0080134, the University of California MICRO Program, Dolby Laboratories, Inc., Lucent Technologies, Inc., Mindspeed Technologies, Inc., and Qualcomm, Inc.

ing indexing and retrieval of high dimensional data sets [19, 21, 23, 32, 44]. However, in those works, VQ has always been used for reducing the number of retrieved feature vectors, and to the best of our knowledge, never for the purpose of efficiently reducing the information retrieved about each data object. In this paper, we propose to design a vector quantizer for the feature space, and store on disk the bit descriptions (indices) of the feature vectors. The number of bits used for each vector is considerably smaller than that used by the uncompressed feature vector description. In order to process the query, only these small number of bits are retrieved, and used to reconstruct the approximate feature vectors via look-up tables that are stored in the computer memory. The reconstructed (approximate) feature vectors are used for distance calculations.

It is often desirable to provide the users with a spectrum of points to choose from on the time/accuracy plane. Such flexibility is needed when diverse users differ in their time constraints and precision requirements, and may trade time for better quality. To accommodate this property, we use a structure called multi-stage VQ (MSVQ). We also use a structure called the split-VQ technique, which, together with the MSVQ structure, allows us to substantially reduce the memory requirements of the look-up tables.

The organization of the paper as follows: In the next section, we discuss related work. In Section 3, the setup phase and the working principles of the proposed algorithm are described. Section 4 describes the clustering of the query distribution. In Section 5, we describe in detail the vector quantization schemes used in our algorithm. Section 6 demonstrates the superiority of VQ-index with experimental analysis. We conclude the paper in Section 7 with a discussion.

2. BACKGROUND AND PREVIOUS WORK

An important goal to be considered in designing a structure for similarity searching in large multi-dimensional data is to minimize the I/O cost during query processing. In this section, we review approaches that have been proposed to overcome the I/O bottleneck in multi-dimensional nearest neighbor queries. We then discuss some techniques in the literature that support approximate nearest neighbor queries.

2.1 Exact NN Query Processing

For efficient query processing in large data sets, it is necessary to build an index structure that reduces the size of the retrieved set needed to answer a query. The general approach is to prune the search space and eliminate irrelevant data objects without accessing the corresponding pages. This approach is referred to as *retrieved set reduction* [19]. Multi-dimensional index trees are very effective in low dimensions. Therefore, they are widely used in low dimensional applications such as Geographical Information Systems (GIS) [13]. However, there are important applications, e.g., multimedia databases [17, 41], that require high dimensional support. It has been observed that the query performance of multi-dimensional index tree structures degrades rapidly with increase in feature vector dimensionality [9, 7, 46, 10]. To overcome the high dimensionality problem, two major approaches were proposed in the literature: dimensionality reduction and scalar quantization. We refer to these approaches as *retrieved information reduction*. In retrieved information reduction approach, the data set is organized such that only a partial representation (e.g., 2 out of d dimensions in dimensionality reduction) of each object is examined.

Index structures: Several index structures have been proposed for retrieval of multidimensional data. Examples include kdb-trees [39], hB-tree [34], R-tree [24], R*-tree [4], SS-tree [47], TV-tree [33], X-tree [9], Pyramid Technique [8], Hybrid Tree [12]. Various algorithms for similarity searching have been developed in conjunction with these indexing mechanisms. The techniques developed for these structures mostly focus on finding the exact result for a query.

A branch-and-bound technique for $k-$NN queries on index tree structures, such as R-tree, was proposed in [40]. Hjaltason and Samet proposed an incremental nearest neighbor (NN) searching algorithm [26] and later adapted it to R-trees [25]. In these techniques, the tree, which consists of minimum bounding rectangles (MBR), is traversed and MBRs are pruned if they are guaranteed not to contain the nearest data point(s).

Dimensionality Reduction: The simultaneous consideration of all the dimensions dramatically degrades the efficiency of high dimensional query processing. A well-studied solution to this problem is the dimensionality reduction approach which is a typical example of retrieved information reduction. Several researchers have used dimensionality reduction for scalable query performance [27, 33, 37, 48, 30]. For example, the dimensions of the feature vectors can be reduced to a desired value such that the underlying indexing technique performs more effectively. There is a trade-off between the accuracy obtained from the information stored in the index structure and the efficiency. The most common dimensionality reduction approach in the literature is based on transformations such as the Discrete Fourier Transform (DFT) [38], the Discrete Cosine Transform (DCT) [29], the Discrete Wavelet Transform (DWT) [11], and Karhunen Loeve Transformation (KLT) [28]. Since these transforms are *distance-preserving*, if a point is in ϵ neighborhood of the query point, it remains in its ϵ neighborhood after the transformation. The high dimensional original feature vectors are transformed, and the resulting vectors are truncated to the first few transform coefficients [1]. The distance calculations are performed in this truncated transform domain, and hence the feature vectors are declared to be closer to the query point than they actually are. This results in extra data in the returned result, and such false hits are later on eliminated by checking the original distances.

Scalar Quantization: An alternative approach for retrieved information reduction is to reduce the number of bits that represent the data objects. Rather than reduce the number of dimensions, which is a crude form of compression, this approach directly compresses the representation to a small number of bits. VA-file [46] is an efficient technique that follows this approach by using a simple scalar quantization. The VA-file is basically a sequential list of approximations of feature vectors, based on independent quantization of the dimensions of the original feature vectors. The total size of the feature-vector set is thereby significantly reduced. For exact nearest neighbor searching, the set of all vector approximations is scanned sequentially and lower and upper bounds on the distance of each vector to the query vector is computed. A significant subset of the vectors is eliminated based on these bounds. Then, the real feature vectors of the candidates are visited to determine the actual k-NN. Recently, an extension called VA$^+$-file [18] has been proposed to handle non-uniform and clustered data sets. In this approach, the creation of a VA-file is improved by first transforming the data using KLT into an "energy compacting" domain. Available bits are non-uniformly allocated to the different dimensions and a quantizer is used to exploit knowledge of the data statistics. These steps result in improved performance, especially for non-uniform data sets. Berchtold et al. have proposed using VA-files with a partition based index structure [6]. They apply a number of VA-files on partitions obtained from a bulk-loading algorithm. The advantage of this technique over standard VA-files is due to pruning of empty spaces by local quantizations instead of global quantization. These local quantizations are performed in the same manner as in the standard VA-file approach.

2.2 Approximate NN Query Processing

Problems due to the curse of dimensionality in high dimensions and scalability problems for very large data sets can be mitigated by structures that support approximate searching. An effective approximate searching technique should organize the data such that acceptable accuracy is achieved while the I/O cost for a query is minimized. Most of the techniques herein had originally been proposed for exact searching, but have recently been extended for approximate searching.

An obvious and simple approach to implement approximate searching is to sequentially scan a portion of the data set. The basic idea is to access the data set progressively while answering the query based on the data that was read. Specifically, data pages are read in the order they are stored and k nearest neighbors can be computed within the data subset that has been read so far. The search can be interactive, and may be stopped any time, to produce the answer available at that point. Accuracy of intermediate answer sets can be improved by storing the data set in a more appropriate manner. Data points that are more likely to be accessed can be stored earlier in the sequential file so that they will be reached earlier in the search process. The corresponding probabilities may be computed using a query history sample, or the data set itself if no such history is available. For each point p, the set of query points that have p in their k-NN neighborhood is computed. Note that this procedure itself is equivalent to reverse nearest neighbor queries (RNN), which were proposed as a new query type that has several applications [31]. The points that have higher number of RNNs are stored earlier in the sequential file.

Index structures: Most of the proposed techniques for approximate nearest neighbor specifically focus on ϵ-nearest neighbor (ϵ-NN) queries. An ϵ-NN is a neighbor of the query point within a factor of $(1+\epsilon)$ of the distance to the true nearest neighbor. In [14], an algorithm was proposed in which the error bound ϵ can be exceeded with a certain probability δ given prior information on the distance distribution of the query point. This technique can be classified as a retrieved set reduction approach. The retrieved data set size is first reduced by the underlying index structure. Besides the irrelevant objects pruned by the index, more objects are pruned by shrinking the NN query sphere at the expense of allowing some false dismissals. The technique also adds a second level of approximation, where more objects can be pruned by allowing to exceed ϵ with the probability δ. In [22], a locality sensitive hashing structure is created by a randomized procedure and the $(1+\epsilon)$-approximate NN point is found with a constant probability. Hashing is used to reduce the retrieved set size, therefore improving query time, again at the expense of false dismissals. This approach is also based on the retrieved set reduction idea, where hashing is used instead of a multi-dimensional index tree to identify the retrieved data set. The disadvantage of this approach, however, is that ϵ needs to be known in advance, and some preprocessing, which is exponential in $1/\epsilon$, is needed. In [49], approximate similarity retrieval with M-trees has been proposed. In [21, 23, 32], retrieved set reduction is achieved by clustering the data, and for each query, retrieving only a few clusters which are stored sequentially in the disk.

Dimensionality Reduction: In [16] a dimensionality reduction approach is proposed for approximate searching. The majority of the dimensionality reduction techniques in the literature satisfy the lower-bound filtering rule to avoid false dismissals [42]. However, [16] intentionally ignores this rule and concentrates only on close approximation of distances in the reduced dimensional domain. The results in the paper establish that high accuracy can be reached when approximate answers are allowed. The use of approximate KLT for dimensionality reduction was proposed in [30], and was shown to be more effective than techniques based on exact transformations, for dynamic data sets.

Scalar Quantization: Weber and Bohm [45] have proposed an approximate NN searching technique based on VA-files. To overcome the I/O bottleneck which is crucial in large databases, they proposed a VA-file based technique which omits the second step of the exact NN search algorithm. The similarity distances are estimated from the lower and upper bounds in the first step, and the result set is created using only these bounds. This introduces some errors in the result set, but yields order of a magnitude speedups over exact NN search. The same approach is applied to VA$^+$-file in [19].

There are several other works on approximate searching. Arya et al. proposed an optimal nearest neighbor algorithm for data structures that are stored in the main memory [3]. An approximate range searching algorithm was proposed in [2]. A more recent technique uses clustering in the KLT domain for approximate nearest neighbor searching [19]. This clustering-based technique outperforms current techniques and achieves significant speedups with accurate results. The results in [19] also show that even simple techniques can achieve acceptable accuracy at significantly reduced time relative to exact searching.

3. A VQ-BASED STRUCTURE FOR EFFICIENT APPROXIMATE K-NN SEARCHING

During the processing of a k-NN query in a high dimensional data set, even if a reduced set is retrieved, if the retrieved information per entry is not reduced, i.e., if the feature vectors are not compressed, then the system has to access many pages. Conversely, if all the database is accessed, no efficient compression scheme will sufficiently reduce the I/O cost. Our premise here is that the solution lies in an appropriate combination of both types of reduction in order to get accurate results with very few pages accessed from the disk. We propose an algorithm based on this premise.

For the retrieved set reduction part, we separate the feature vectors into subsets, whose union is the entire data set; retrieved information reduction is achieved by VQ, as discussed in detail in Sections 4 and 5, respectively. Quantization refers to partitioning the space into regions, and assigning to each region a representative, so that data points are mapped to the representative of the region they fall into. In scalar quantization, each dimension is quantized separately, hence the resulting regions are rectangular hyperboxes that tile the data space (See Figure 1 for a simple two dimensional example.) Because of the geometric simplicity of the regions, the scalar quantizer is easy to maintain, and hence was successfully used by the VA-file technique [46] for exact k-NN queries. However, scalar quantization is potentially a very suboptimal partition of the space, because it ignores the correlation between components. On the other hand, as seen in Figure 1, the resulting regions of a general VQ are convex polytopes which represent the data more accurately. The regions are much harder to maintain, but for our pur-

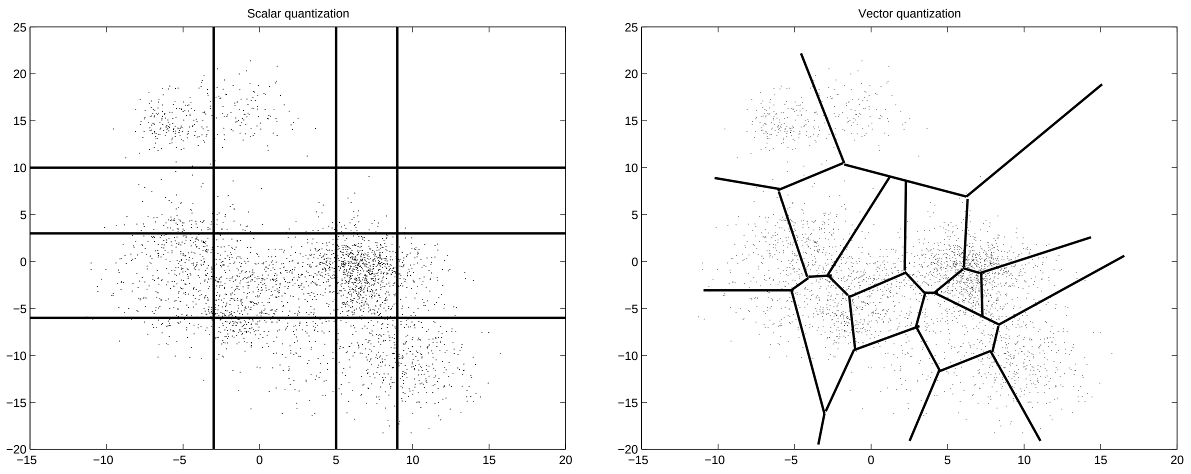


Figure 1: Resultant 16 regions in scalar (left) and vector (right) quantization

poses, it suffices to keep the representatives of the regions, not the regions themselves.

In Figure 2, the overall system setup is displayed as a block diagram. Using the query history, the query space is partitioned into clusters, and the resulting query clusters induce a partitioning of the data set. Each data subset is then efficiently compressed using a vector quantizer which is tailored to specific subset, and the resulting bit descriptions are written into separate files.

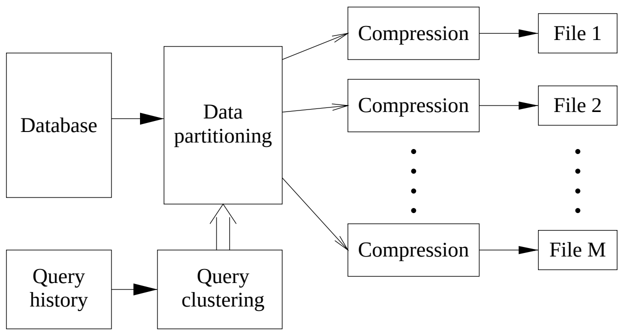


Figure 2: Block diagram of the setup phase of the system

As shown in Figure 3, upon receiving a query, the system accesses only the *compressed* information about *one* chosen subset of data. The algorithm first decides which Voronoi cell the given query resides in, and only the corresponding file is retrieved from the disk. The distance calculations, for the purpose of finding the k nearest neighbors, are then performed between the query vector and the decompressed (or reconstructed) feature vectors retrieved from the chosen file.

4. RETRIEVED SET REDUCTION VIA DATA PARTITIONING

In this section we discuss how the retrieved set reduction is accomplished. We use the accumulated query history to partition the data. If no such history exists, we use the data set itself for that purpose. A random subset is sampled

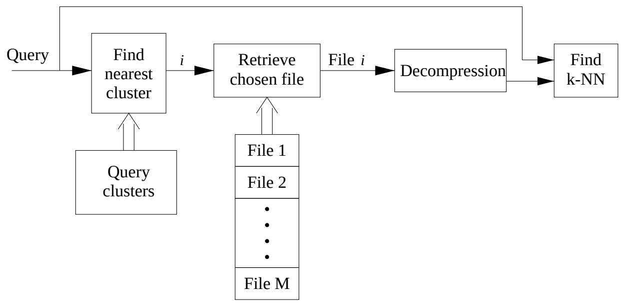


Figure 3: Block diagram describing the working principle of the system

from the query history, since the history might be too large to handle (this is obviously the case if we take the data set itself as the history). For the sake of faithful statistical characterization, however, we keep the number of elements in the sample query set large enough. We then cluster the sample query set into M clusters. For this purpose, we use the K-means algorithm [15], because the intuition behind its iterations match our main objectives. Let $\mathcal{V}_i$ denote the resultant *Voronoi* cell for the cluster i. For each point $\mathbf{q}$ in the sample query set, let us denote by $\mathcal{N}_L(\mathbf{q})$ the points in the data set which are the L nearest neighbors of $\mathbf{q}$. We create the data subset $\mathcal{S}_i$ as follows:

$$\mathcal{S}_i = \bigcup_{\mathbf{q}\in\mathcal{V}_i} \mathcal{N}_L(\mathbf{q}) \; . \tag{1}$$

Assuming that the query sample is statistically representative, if an actual query falls into $\mathcal{V}_i$, there is a high probability that its nearest neighbors are in the subset $\mathcal{S}_i$. Note that the same data point may fall into more than one subset, and some data points may not be covered, i.e., may not fall into any $\mathcal{S}_i$. (This only happens if the data point is not in the L-nearest neighborhood of any query $\mathbf{q}$.) Since uncovered data points are statistically among the least likely to be in the answer set, adopting a simple strategy, we investigate which Voronoi cell $\mathcal{V}_i$ the uncovered point belongs to, and then include it in the corresponding subset $\mathcal{S}_i$.

The choice of M and L is subject to the following considerations:

- If M is too small, then little will be achieved in terms of retrieved set reduction. If M is too large, however, the amount of repetition of the same data point in several $\mathcal{S}_i$ will be amplified and hence the efficiency in retrieved set reduction will be compromised. Moreover, since the look-up tables for each subset for feature vector reconstruction are stored in the memory, the required memory size will increase.
- If L is too small, then many true nearest neighbors to the given query will fall into subsets other than the chosen one and, hence, the accuracy of the approximate k-NN result will be poor. Moreover, many points will remain uncovered after the initial partitioning, which will also result in decreased accuracy. If L is too large, on the other hand, then data repetition will be excessive. For k-NN search queries, k is usually below a prespecified number, $k_{\max}$, according to user expectations, and the readability of the result. Our experiments show that an appropriate range for L is 5 to 10 times $k_{\max}$.

5. VQ FOR RETRIEVED INFORMATION REDUCTION

A vector quantizer $\mathcal{Q}$ of dimension d and size N is a mapping from a vector in d-dimensional Euclidean space, $\mathcal{R}^d$, into a finite set $\mathcal{C}$ containing N reproduction vectors (also called *codevectors*.) The quantizer can be decomposed into two operations, the *encoder*, and the *decoder*. The encoder $\mathcal{E}$ is the mapping from $\mathcal{R}^d$ to the set $\mathcal{J} = \{1, 2, \ldots, N\}$, and the decoder $\mathcal{D}$ maps the set $\mathcal{J}$ into the reproduction set $\mathcal{C}$ (also called the *codebook*, or the *look-up table*.)

For each feature vector subset $\mathcal{S}_i$, we design a vector quantizer $\mathcal{Q}_i$, and store the corresponding codebook

$$\mathcal{C}_i = \{\hat{\mathbf{x}}_{i_1}, \hat{\mathbf{x}}_{i_2}, \ldots, \hat{\mathbf{x}}_{i_{N_i}}\}$$

in the memory. For each feature vector $\mathbf{x} \in \mathcal{S}_i$, we write into disk the encoder output $\mathcal{E}_i(\mathbf{x})$ sequentially, and in binary notation. When the subset $\mathcal{S}_i$ is chosen to be retrieved for query processing, the algorithm retrieves the bit encoding of all $\mathbf{x} \in \mathcal{S}_i$, and then using the codebook $\mathcal{C}_i$, decodes $\hat{\mathbf{x}} = \mathcal{Q}_i(\mathbf{x}) = \mathcal{D}_i(\mathcal{E}_i(\mathbf{x}))$ for each $\mathbf{x}$. These reconstructed values are then used for calculating the distance between the query point and the feature vectors. See Figure 4 for a pictorial description.

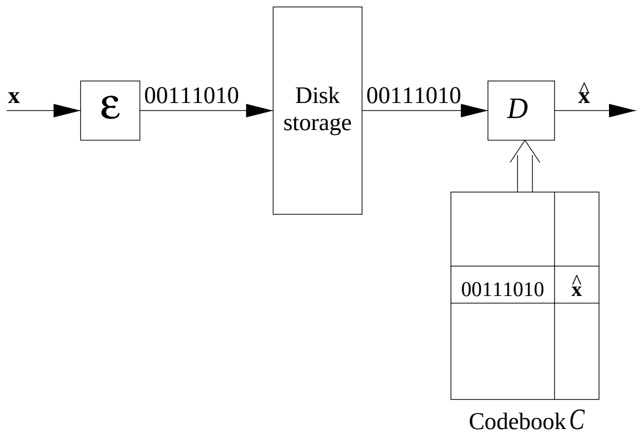


Figure 4: A vector quantizer

Vector quantization is in fact theoretically the "optimal" solution for fixed rate compression of a vector set [20]. Although at first glance, the boldness of this statement may raise some skepticism, it is easy to verify. Any compression scheme that maps a vector into one of N binary words and reconstructs the approximate vector from the binary word, no matter how complicated the encoding/decoding may be, is trivially implementable by a VQ with a codebook of N codevectors.

Because of limitations on the computer memory, codebook sizes, N_i, must be much smaller than the number of feature vectors in the corresponding $\mathcal{S}_i$. Hence, the reconstruction $\hat{\mathbf{x}}$ is only an approximation to $\mathbf{x}$. Obviously, higher accuracy in feature vector reconstruction yields better approximation of the distance between the query point and the feature vectors. The objective of the design of each quantizer $\mathcal{Q}_i$ is therefore

$$\min_{\mathcal{Q}_i} \sum_{\mathbf{x} \in \mathcal{S}_i} d(\mathbf{x}, \mathcal{Q}_i(\mathbf{x})) ,$$

where $d(\cdot,\cdot)$ is the distance or distortion function used for measuring similarity. If we express the objective explicitly in terms of the encoder and the decoder functions, we get

$$\min_{\mathcal{D}_i} \min_{\mathcal{E}_i} \sum_{\mathbf{x} \in \mathcal{S}_i} d(\mathbf{x}, \mathcal{D}_i(\mathcal{E}_i(\mathbf{x}))) . \tag{2}$$

The objective may also be expressed as

$$\min_{\mathcal{E}_i} \min_{\mathcal{D}_i} \sum_{j=1}^{N_i} \sum_{\mathbf{x}:\mathcal{E}_i(\mathbf{x})=j} d(\mathbf{x}, \mathcal{D}_i(j)) . \tag{3}$$

Equations (2) and (3) help us determine the best encoder for a fixed decoder, and the best decoder for a fixed encoder, respectively. Indeed, if we fix the decoder $\mathcal{D}_i$, and hence the codebook $\mathcal{C}_i$, it follows from (2) that the best encoder is given by

$$\mathcal{E}_i^*(\mathbf{x}) = \arg\min_j d(\mathbf{x}, \hat{\mathbf{x}}_{ij}) . \tag{4}$$

If we fix the encoder $\mathcal{E}_i$, then from (3), the best decoder becomes

$$\mathcal{D}_i^*(j) = \arg\min_{\hat{\mathbf{x}}} \sum_{\mathbf{x}:\mathcal{E}_i(\mathbf{x})=j} d(\mathbf{x}, \hat{\mathbf{x}}) . \tag{5}$$

Equations (4) and (5) can be interpreted as follows: When the codebook is fixed, the best codevector as an approximation for the given vector $\mathbf{x}$ is the nearest one in the codebook. Similarly, when a group of vectors are assigned to the same codevector, the best value for that codevector is the one minimizing its average distance to all vectors in the group. In the important case of the squared Euclidean distance (also referred to as "mean squared error"), the best update becomes

$$\mathcal{D}_i^*(j) = \text{mean}\{\mathbf{x} \mid \mathcal{E}_i(\mathbf{x}) = j\} . \tag{6}$$

Iteration of (4) and (6) until convergence is what is commonly known as the generalized Lloyd's algorithm in the VQ literature [20]. The algorithm is guaranteed to converge, but it requires a good initialization for convergence to a good solution. A good heuristics is to design the codebook initially for only a small number of codevectors, say 1, and then to grow the codebook by each time duplicating and perturbing a chosen codevector, and reiterating the Lloyd's algorithm until convergence.

5.1 Structurally Constrained VQ

Since we store the codebooks $\mathcal{C}_i$ for each file in the computer memory, there is a limitation on their sizes. For example, assume file i has 65536 feature vectors stored in it.

If we decide to use a codebook of size 65536, the best codebook then consists of the feature vectors themselves. The resultant VQ is very efficient in the sense that by retrieving only 16 bits per vector from the disk (instead of $32d$, where d is the dimensionality, and 32 is the number of bits in floating point representation), one can access the feature vectors with zero error. However, in most cases, the memory is much smaller in size (otherwise, there would be no need for any disk storage in the first place).

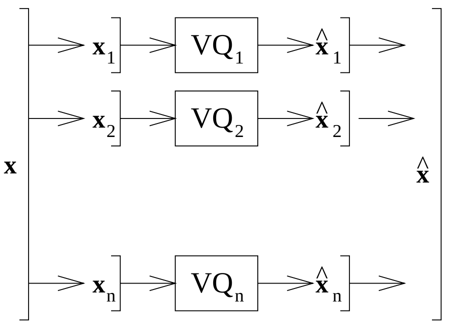


Figure 5: Split-VQ algorithm

One approach to mitigate the memory storage barrier is to impose certain structural constraints on the codebook [20]. This means that the codevectors cannot have arbitrary values but are distributed in a restricted manner. The simplest example is the split-VQ structure, where the vectors are split into a few subvectors, and each subvector is quantized separately (See Figure 5). For the above example, we can split the vectors into two parts with equal dimensionality, and quantize each part with 8 bits. As a result, we still obtain a quantizer with $256 \times 256 = 65536$ codevectors, however, the codebook is a Cartesian product of two codebooks of dimensionality $d/2$ and size 256, and hence the actual degree of freedom for the codevectors is $256d$, instead of $65536d$. Although this constraint causes degradation to the performance of the quantizer, it also results in a major (factor of 256) savings in terms of codebook storage. We can split the vector into more parts in order to further reduce the required memory storage size. As an extreme example, we can split each vector into 16 parts and design a codebook of size 2 for each subvector of dimensionality $d/16$, and hence reduce the codebook storage down to $2d$, while keeping the number of possible vector reconstructions equal to 65536. Obviously the complexity-quality trade-off here is due to the fact that as we split the vector into more and more subvectors, since we quantize each subvector independently, we further and further disregard the *correlation* between the dimensions, which could have been exploited for reproduction quality.

Another structurally constrained VQ which is used for codebook storage reduction is known as the multi-stage VQ (MSVQ). As shown in Figure 6, there are several stages, each of which quantizes the previous quantization error, i.e., the *residual* vector from the previous stage. For the same example as above, if we use 2 stages, and design codebooks of dimensionality d and size 256, we again obtain 65536 possible codevectors, but this time the codebook storage is reduced to $512d$, instead of $256d$ as in the split-VQ case. The compensation for this less extreme storage reduction is that at each stage quantized vectors are of full dimensionality, hence the correlation between dimensions is fully exploited.

The MSVQ structure also provides progressive improvement in the quality of the reconstructed vectors. The reconstructed values $\hat{\mathbf{x}}_j$ in Figure 6 gradually approach the unquantized value $\mathbf{x}$, as j is increased. This is a very desirable feature, since the users might differ in their time constraints, i.e., some users might wish to get *some* approximate answer in a very short amount of time, while others might trade time for better quality.

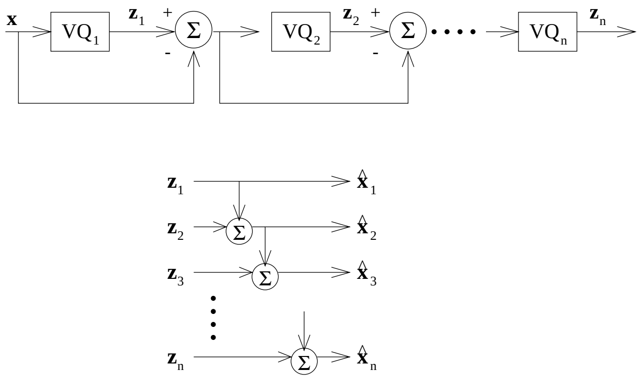


Figure 6: Multi-stage VQ algorithm. The quantized values $\hat{\mathbf{x}}_j$ are progressively refined, as j increases.

In our algorithm, we employ both the MSVQ and the split-VQ structures in the same compression scheme, as depicted in Figure 7. The design objective of such a system is to strike a good balance between the number of stages and the number of subvectors.

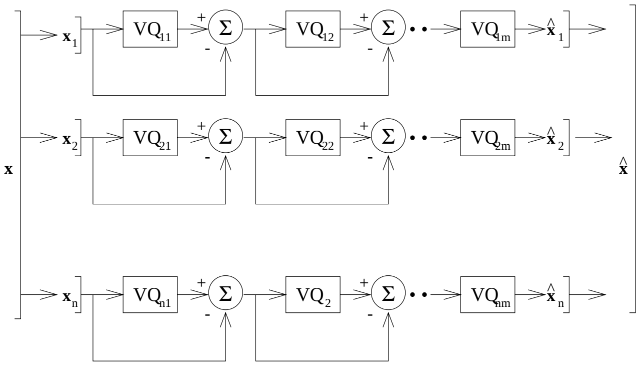


Figure 7: Split-MSVQ algorithm.

6. EXPERIMENTAL RESULTS

In this section, we provide a performance analysis of the proposed and current state-of-the-art techniques. We first discuss the basis for performance comparisons between any two approximate k-NN searching techniques. The superiority of the proposed algorithm is then demonstrated using real-world experimental setups from different application domains. The first data set, *Satellite Image Texture* (LANDSAT), is of size 100,000 with 60-dimensional feature vectors extracted from satellite images [36]. This data set is widely used in high dimensional indexing and similarity searching research [35, 22, 14]. The second data set, HISTOGRAM, is a color image histogram data set of size 10,000 and dimensionality 64. The vectors represent the frequency of occurrences of colors in each of the 64 sub-cubes in the Red-Green-Blue color cube.

6.1 The Basis for Performance Comparisons for Approximate k-NN Searching

For similarity queries, the quality of the result set is usually measured by two quantities; *recall* and *precision* [40]. Recall is a measure of the completeness of the retrieved set, i.e., the percentage of the retrieved data in the exact answer set. Precision, on the other hand, measures the purity of the retrieved set, i.e., the percentage of relevant objects in the answer set. The irrelevant objects in the result set are called *false hits* and the relevant objects that are not in the answer set are called *false dismissals*. The traditional measures precision and recall disregard the fact that the extraction of feature vectors is already based on heuristics and, hence, that the feature vectors represent an approximation of the real data. These measures are too harsh in their rewarding and penalizing policies. For example, in Figure 8, the two nearest neighbors of the query point q are points a and b. The points c and d are also close to the query point and are potentially interesting, whereas points e and f are considerably less likely to interest the user. However, if approximation techniques $\mathcal{A}$ and $\mathcal{B}$ return $\{c, d\}$ and $\{e, f\}$, respectively, precision and recall measures will yield 0 for both techniques, and hence will not differentiate their relative merits.

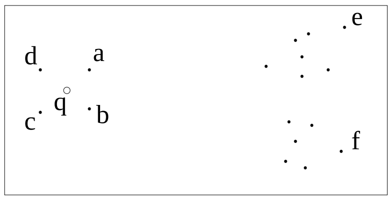


Figure 8: Harshness of recall and precision measures.

In [45], a measure involving the rankings of the elements in the answer set in terms of closeness to the query point was considered. Figure 9 intends to demonstrate how misleading this measure can be. For example, points g and h are the third and the fourth nearest neighbors to the query point. According to any ranking-based measure, therefore, $\{g, h\}$ is considered a fairly good answer set, although points g and h are likely to be uninteresting because of their distance to the query.

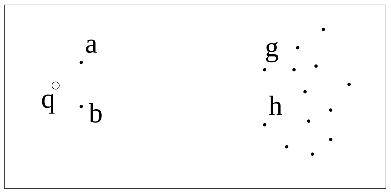


Figure 9: Inaccuracy of ranking-based measures.

An alternative quality measure was introduced in [19], and [49]. We now introduce a slightly more generalized version of that measure, to be used in our performance comparisons of various approximate k-NN searching methods. Suppose the approximate k-NN searching algorithm $\mathcal{A}$ returns the result set $\{\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_{k'}\}$ and the actual (or golden) result set according to the underlying distance function $d(\cdot, \cdot)$ is $\{\mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_k\}$, where $k' \geq k$. We define the error metric D as the ratio of the average distance of each set to the query point, i.e.,

$$D_{\mathcal{A}}(\mathbf{q}) = \frac{\frac{1}{k'} \sum_{i=1}^{k'} d(\mathbf{q}, \mathbf{a}_i)}{\frac{1}{k} \sum_{i=1}^{k} d(\mathbf{q}, \mathbf{g}_i)} \qquad (7)$$

Note that since $k' \geq k$, it is always true that $D \geq 1$, where equality is satisfied if the approximate answer is equal to the exact answer. The reason we consider the extra case $k' > k$ is because there might be a tie for the k'th nearest neighbor, in which case we assume the algorithm is allowed to output all vectors having the same *measured* distance to the query point. In general, this phenomenon may occur for any compression-based scheme, because more than one point may be reconstructed as the same vector, and hence assume the same rank with respect to closeness to the query point.

6.2 Performance Comparisons

For both data sets, we compare the performance of the proposed method with other approximation techniques, for different values of k. The comparison is based on the number of pages accessed in order to achieve results with prescribed acceptable accuracy levels, i.e., given small values of D. Other approximation techniques used for comparison are, VA-file [46, 45], VA$^+$-file, and KLT-Domain Clustering [18, 19]. These techniques have been shown to be very effective specifically for high dimensional databases, and to outperform earlier techniques. We refer to our method as *VQ-index*.

The capacity of one disk page is taken as 1Kbytes for the HISTOGRAM set, and 8Kbytes for the LANDSAT set, since the latter set is larger. These page capacities correspond to 256 and 2048 floating-point numbers, respectively. The whole data set fits into 2500 pages for the HISTOGRAM set, and 2930 pages for the LANDSAT set.

In Figure 10, we display the comparison of number of accessed pages for $k = 10$ and $k = 50$, for the data set LANDSAT. For $k = 10$, we observe a speedup between 1.8 and 2.4 over the second best method, which is the KLT-Domain Clustering algorithm, between 10 and 13 over the VA$^+$-file method, and between 19 and 23 over the standard VA-file algorithm. For $k = 50$, the speedups raise to between 2.5 and 3.5 over the KLT-Domain Clustering, between 11 and 14 over the VA$^+$-file, and between 21 and 25 over the VA-file methods. The reason for the better performance for a higher value of k is that the effect of the ties mentioned in the previous section alleviate as k increases. These performance values are obtained when we use in our experiments the same sample (training) query set extracted from the query history as our actual query set. In Figure 11, we demonstrate the effects of using a different (test) query set extracted from the same distribution. As seen from the figure, the resultant performance deterioration is virtually negligible, and the speedups mentioned above prevail. Passing this test indicates that we overcome the phenomenon known in the clustering literature as *overtraining*. Overtraining refers to the case where the design method adapts itself to the training set in an excessive manner to the extent that the performance significantly degrades when the system is tested on real data independent of the training sample.

In the experiments with the LANDSAT data set, we set the number of files to $M = 32$, and during the data partitioning, we take into subset $\mathcal{S}_i$ all the points within $L = 250$ neighborhood of each sample query point falling into Voronoi cell $\mathcal{V}_i$. The sample query set is of size 10,000 and is extracted randomly from the data set. After the data partitioning process, the total number of elements of all the subsets is around 400,000. However, since only the compressed

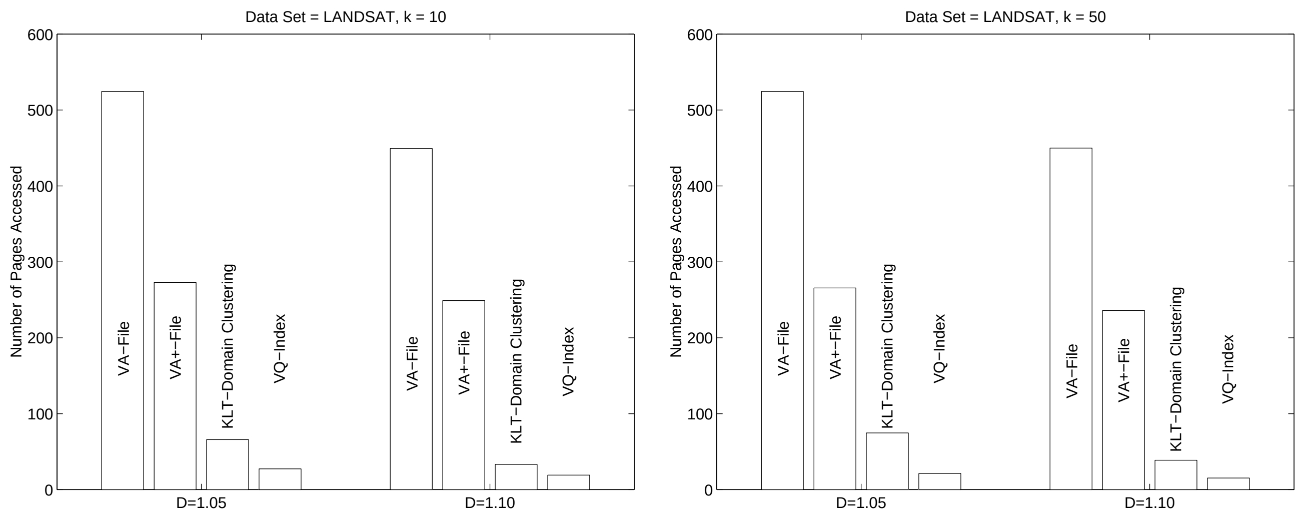


Figure 10: Performance comparison for the LANDSAT data set at k=10 and k=50.

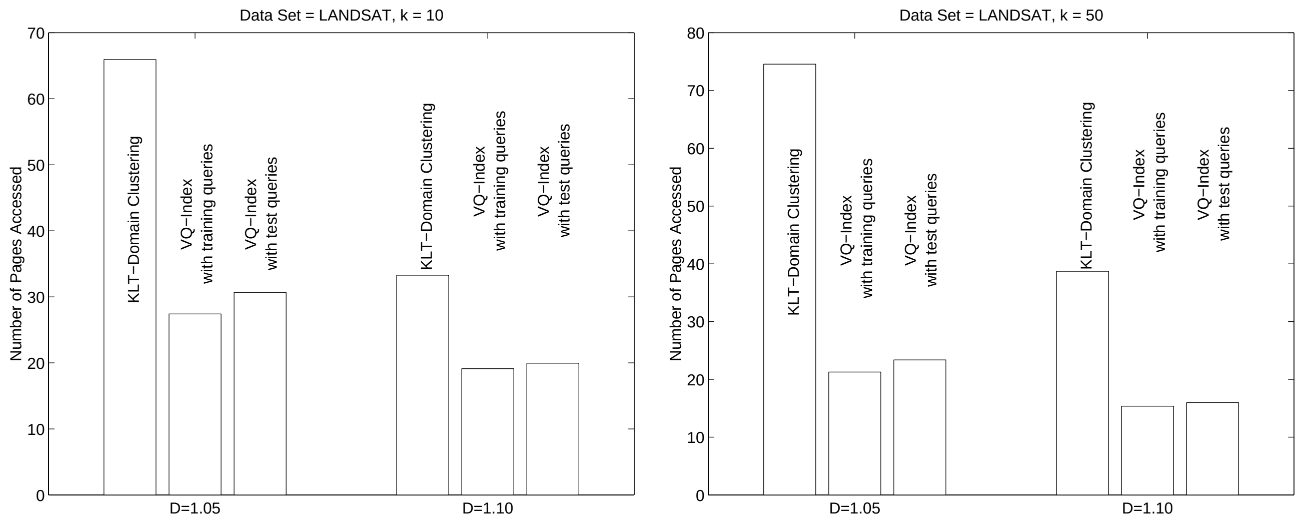


Figure 11: The effects of testing the algorithm outside the query training set.

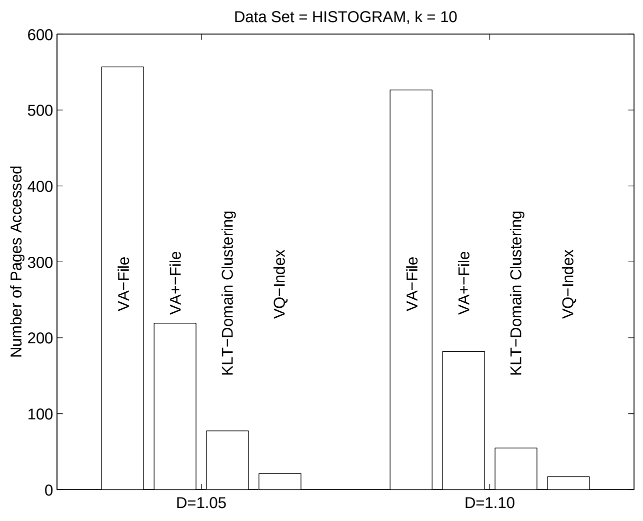
values are written to disk, the actual disk storage becomes about 5 times smaller than the uncompressed original database size. The memory requirement for the codebooks is about 5% of the database size, i.e., the total number of 60-dimensional entries in the look-up tables is around 5,000. If the memory specifications are so severe that even this percentage of the database size is not tolerated, then each look-up table can be also stored in the corresponding file. Hence, upon deciding which file to extract, the system can first retrieve the codebooks, and then the compressed feature vectors. The overhead of such a scenario is only about 6 pages on the average.

In Figure 12, we display the corresponding performance comparison results for the data set HISTOGRAM, at $k = 10$ and $k = 20$. For $k = 10$, we observe a speedup of 3.2 to 3.65 over the KLT-Domain Clustering, 10 over the VA$^+$-file, and 26 to 30 over the VA-file methods. For $k = 20$, the speedups raise to between 3.75 and 4.5 over the KLT-Domain Clustering, 11 over the VA$^+$-file, and between 28 and 36 over the VA-file methods. For this experiment we set $M = 8$, and $L = 100$. The sample query set is of size 1,000 and is again extracted from the data set itself. After the data partitioning, the total number of elements in all the subsets rise to about 19,000. The actual disk storage after the compression is, however, less than 6% of the original database size. For codebook storage, a memory of 5% of the database size is needed.

7. CONCLUSION AND DISCUSSION

We introduced *VQ-index*, a novel approach for indexing and efficient approximate nearest neighbor searching based on optimal compression of the feature space. An efficient approximate k-NN searching algorithm must reduce not only the number of vectors retrieved from disk, but also the information retrieved about each vector. The novelty of algorithm is in how both tasks are achieved.

In order to achieve a good retrieved set reduction, we use a training set of query samples extracted from the query history. In the absence of any query history, one way is to use the data distribution itself. We cluster the sample query set



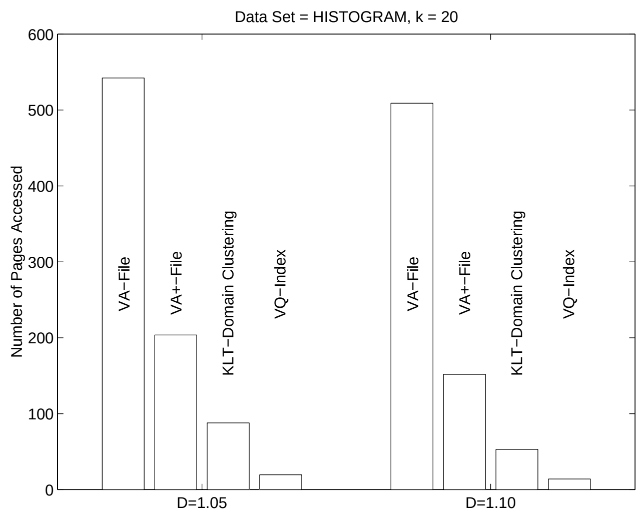


Figure 12: Performance comparison for the HISTOGRAM data set for k=10 and k=20.

using the K-means algorithm, and split the data into subsets using a simple rule based on a statistical observation; if an actual query falls into a fixed Voronoi cell, then its nearest neighbors in the data set are most probably also among a larger set of nearest neighbors of sample queries in the same Voronoi cell. Of course, this observation relies on a fine statistical sampling of the query distribution. Otherwise, the resultant data partitioning may be "overtrained" to the sample query set, and the approximate nearest neighbor searching performance may degrade. A common validation in the clustering literature is to evaluate the performance of the algorithm using a test set extracted from the same distribution as the training set. The proposed algorithm passed this test.

For the purpose of reducing the retrieved information, we utilized the most efficient compression scheme, namely vector quantization. Previously considered compression methods [45, 18] adopt scalar quantization, which is potentially a very suboptimal special case of VQ. The suboptimality follows from the fact that the correlation between components is ignored. Existing dimensionality reduction techniques can also be considered as special cases of scalar quantization (suppressed components are quantized to 0 bits, and the remaining degenerate components are quantized to 32 bits since they are represented as floating point numbers.) For each subset of data, we designed separate vector quantizers tailored to the distribution of the vectors in the subset. The compressed representation of each subset is then stored in the disk as separate files. Hence, the proposed algorithm achieves major reduction in disk I/Os, as in order to answer a query with acceptable accuracy, only quantization indices for the corresponding subset of feature vectors are retrieved, followed by calculation of approximate distances based on their reconstruction.

Since the look-up tables for reconstructed vector generation reside in the memory, we use structurally constrained VQ techniques, namely, multi-stage VQ and split-VQ, to reduce the required memory. The multi-stage VQ technique inherently enables successive refinement of the answer set accuracy as additional disk I/Os are made. It hence offers the flexibility to trade time for better quality. This is a very desirable feature, since users may differ in their time constraints.

Our experiments demonstrate significant speedups over other approximate nearest neighbor searching techniques, when the accuracy of the approximation is fixed at acceptable levels.

8. REFERENCES

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *4th Int. Conference on Foundations of Data Organization and Algorithms*, pages 69–84, 1993.

[2] S. Arya and D. M. Mount. Approximate range searching. In *11th Annual Symposium on Computational Geometry*, pages 172–181, June 1995.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching. In *5th Ann. ACM-SIAM Symposium on Discrete Algorithms*, pages 573–582, 1994.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[5] R. Bellman. *Adaptive Control Processes: A Guided Tour.* Princeton University Press, 1961.

[6] S. Berchtold, C. Bohm, H. Jagadish, H. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. 16th Int. Conf. on Data Engineering*, San Diego, CA, 2000.

[7] S. Berchtold, C. Bohm, D. Keim, and H. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, pages 78–86, Tuscon, Arizona, June 1997.

[8] S. Berchtold, C. Bohm, and H.-P. Kriegel. The Pyramid-Technique: Towards breaking the curse of dimensionality. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 142–153, Seattle, Washington, USA, June 1998.

[9] S. Berchtold, D. Keim, and H. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 28–39, Bombay, India, 1996.

[10] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful. In *Int. Conf. on Database Theory*, pages 217–225, Jerusalem, Israel, January 1999.

[11] K. R. Castleman. *Digital Image Processing.* Prentice-Hall, Inc., 1996.

[12] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In

Proc. Int. Conf. Data Engineering, pages 440–447, Sydney, Australia, 1999.

[13] X. Cheng, R. Dolin, M. Neary, S. Prabhakar, K. Ravikanth, D. Wu, D. Agrawal, A. El Abbadi, M. Freeston, A. Singh, T. Smith, and J. Su. Scalable access within the context of digital libraries. In *IEEE Proceedings of the International Conference on Advances in Digital Libraries, ADL*, pages 70–81, Washington, D.C., 1997.

[14] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In *Proc. Int. Conf. Data Engineering*, pages 244–255, San Diego, California, March 2000.

[15] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

[16] O. Egecioglu and H. Ferhatosmanoglu. Dynamic dimensionality reduction and similarity distance computation by inner product approximations. In *Proceedings of the 9th ACM Int. Conf. on Information and Knowledge Management*, pages 219–226, McLean, Virginia, November 2000.

[17] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3:231–262, 1994.

[18] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Vector approximation based indexing for non-uniform high dimensional data sets. In *Proceedings of the 9th ACM Int. Conf. on Information and Knowledge Management*, pages 202–209, McLean, Virginia, November 2000.

[19] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Approximate nearest neighbor searching in multimedia databases. In *To appear in Proc of 17th IEEE Int. Conf. on Data Engineering (ICDE)*, Heidelberg, Germany, April 2001.

[20] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, MA, 1992.

[21] E. Giladi, M. Walker, J. Wang, and W. Volkmuth. SST: An algorithm for searching sequence databases in time proportional to the logarithm of the database size. In *RECOMB*, Japan, 2000.

[22] A. Gionis, P. Indyk, and R. Motwani. Similarity searching in high dimensions via hashing. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 518–529, Edinburgh, Scotland, UK, September 1999.

[23] K. Goh and E. Chang. Indexing multimedia data in high-dimensional and weighted feature spaces. In *The 6th Visual Database Conference*, Australia, May 2002.

[24] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[25] G. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM Transactions on Database Systems*, 24(2):265–318, 1999.

[26] G. R. Hjaltason and H. Samet. Ranking in spatial databases. In *Proc. of 4th Int. Symp. on Large Spatial Databases*, pages 83–95, Portland,ME, 1995.

[27] D. Hull. Improving text retrieval for the routing problem using latent semantic indexing. In *Proc. of the 17th ACM-SIGIR Conference*, pages 282–291, 1994.

[28] N. S. Jayant and P. Noll. *Digital Coding of Waveforms*. Prentice-Hall, Inc., 1984.

[29] T. Kailath. *Modern Signal Processing*. Springer Verlag, 1985.

[30] K. V. R. Kanth, D. Agrawal, and A. Singh. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 166–176, Seattle, Washington, June 1998.

[31] F. Korn and S. Muthukrishnan. Influence sets based on reverse nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Dallas, USA, May 2000.

[32] C. Li, E. Chang, H. Garcia-Molina, and G. Wiederhold. Clustering for approximate similarity search in high-dimensional spaces. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):792–808, July–August 2002.

[33] K. Lin, H. V. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517–542, 1995.

[34] D. B. Lomet and B. Salzberg. The hb-tree: A multi-attribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.

[35] B. S. Manjunath. Airphoto dataset. http://vivaldi.ece.ucsb.edu/Manjunath/research.htm, May 2000.

[36] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):837–42, August 1996.

[37] W. Niblack, R. Barber, W. Equitz, M. Flickner, E. Glasman, D. Petkovic, and P. Yanker. The QBIC project: Querying images by content using color, texture and shape. In *Proc. of the SPIE Conf. 1908 on Storage and Retrieval for Image and Video Databases*, volume 1908, pages 173–187, February 1993.

[38] A. V. Oppenheim and R. W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, Inc., 1989.

[39] J. T. Robinson. The kdb-tree: A search structure for large multi-dimensional dynamic indexes. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.

[40] N. Roussopoulos, S. Kelly, and F. Vincent. Nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 71–79, San Jose, California, May 1995.

[41] T. Seidl and Kriegel H.-P. Efficient user-adaptable similarity search in large multimedia databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 506–515, Athens, Greece, 1997.

[42] T. Seidl and H.P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Chicago, Illinois, U.S.A., June 1998. ACM.

[43] V.S. Subrahmanian. *Principles of Multimedia Database Systems*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999.

[44] E. Tuncel and K. Rose. Towards optimal clustering for approximate similarity searching. In *Proceedings of IEEE International Conference on Multimedia and Expo*, Lausanne, Switzerland, August 2002.

[45] R. Weber and K. Bohm. Trading quality for time with nearest-neighbor search. In *Proc. Int. Conf. on Extending Database Technology*, pages 21–35, Konstanz, Germany, March 2000.

[46] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 194–205, New York City, New York, August 1998.

[47] D. White and R. Jain. Similarity indexing with the SS-tree. In *Proc. Int. Conf. Data Engineering*, pages 516–523, 1996.

[48] D. Wu, D. Agrawal, A. El Abbadi, and T. R. Smith. Efficient retrieval for browsing large image databases. In *Proc. Conf. on Information and Knowledge Management*, pages 11–18, November 1996.

[49] P. Zezula, P. Savino, G. Amato, and F. Rabitti. Approximate similarity retrieval with M-trees. *The VLDB Journal*, 4:275–293, 1998.