Adaptive Cluster Distance Bounding for High Dimensional Indexing[+]

Sharadh Ramaswamy[*], *Student Member, IEEE,* and Kenneth Rose[†], *Fellow, IEEE*

***Abstract*—We consider approaches for similarity search in correlated, high-dimensional data-sets, which are derived within a clustering framework. We note that indexing by "vector approximation" (VA-File), which was proposed as a technique to combat the "Curse of Dimensionality", employs *scalar quantization*, and hence necessarily ignores dependences across dimensions, which represents a source of suboptimality. Clustering, on the other hand, exploits inter-dimensional correlations and is thus a more compact representation of the data-set. However, existing methods to prune irrelevant clusters are based on bounding hyperspheres and/or bounding rectangles, whose lack of tightness compromises their efficiency in exact nearest neighbor search. We propose a new cluster-adaptive distance bound based on separating hyperplane boundaries of Voronoi clusters to complement our cluster based index. This bound enables efficient spatial filtering, with a relatively small pre-processing storage overhead and is applicable to Euclidean and Mahalanobis similarity measures. Experiments in exact nearest-neighbor set retrieval, conducted on real data-sets, show that our indexing method is scalable with data-set size and data dimensionality and outperforms several recently proposed indexes. Relative to the VA-File, over a wide range of quantization resolutions, it is able to reduce random IO accesses, given (roughly) the same amount of sequential IO operations, by factors reaching 100X and more.**

***Index Terms*—Multimedia databases, indexing methods, similarity measures, clustering, image databases.**

I. Introduction

WITH developments in semiconductor technology and powerful signal processing tools, there has been a proliferation of personal digital media devices such as digital cameras, music and digital video players. In parallel, storage media (both magnetic and optical) have become cheaper to manufacture and correspondingly their capacities have increased. This has spawned new applications such as Multimedia Information Systems, CAD/CAM, Geographical Information systems (GIS), medical imaging, time-series analysis (in stock markets and sensors), that store large amounts of data periodically in and later, retrieve it from databases (see [1] for more details). The size of these databases can range from the relatively small (a few 100 GB) to the very large (several 100 TB, or more). In the future, large organizations will have to retrieve and process petabytes of data, for various purposes such as data mining and decision support. Thus, there exist numerous applications that access large multimedia databases, which need to be effectively supported.

[+] This work was supported in part by NSF, under grants IIS-0329267 and CCF-0728986, the University of California MICRO program, Applied Signal Technology Inc., Cisco Systems Inc., Dolby Laboratories Inc., Sony Ericsson Inc., and Qualcomm Inc.

[*] Mayachitra Inc.; email: rsharadh@mayachitra.com
[†] University of California, Santa Barbara; email: rose@ece.ucsb.edu

Spatial queries, specifically nearest neighbor queries, in high-dimensional spaces have been studied extensively. While several analyses [2], [3], have concluded that the nearest-neighbor search, with Euclidean distance metric, is impractical at high dimensions due to the notorious "curse of dimensionality", others have suggested that this may be overpessimistic. Specifically, the authors of [4] have shown that what determines the search performance (at least for R-tree-like structures) is the *intrinsic* dimensionality of the data set and not the dimensionality of the address space (or the *embedding* dimensionality).

The typical (and often implicit) assumption in many previous studies is that the data is uniformly distributed, with independent attributes. Such data-sets have been shown to exhibit the "curse of dimensionality" in that distance between all pairs of points (in high dimensional spaces) converges to the same value [2], [3]. Clearly, such data-sets are impossible to index as the nearest and furthest neighbors are indistinguishable. However, real data sets overwhelmingly invalidate assumptions of independence and/or uniform distributions; rather, they typically are skewed and exhibit intrinsic (*fractal*) dimensionalities that are much lower than their embedding dimension, e.g., due to subtle dependencies between attributes. Hence, real data-sets are demonstrably indexable with Euclidean distances. Whether the Euclidean distance is *perceptually* acceptable is an entirely different matter, the consideration of which directed significant research activity in content-based image retrieval toward the Mahalanobis (or weighted Euclidean) distance (see [5],[6]). Therefore, in this paper, we focus on real data-sets and compare performance against the state-of-the-art indexing targetting real data-sets.

II. Our Contributions

In this section, we outline our approach to indexing real high-dimensional data-sets. We focus on the clustering paradigm for search and retrieval. The data-set is clustered, so that clusters can be retrieved in decreasing order of their probability of containing entries relevant to the query.

We note that the Vector Approximation (VA)-file technique [7] implicitly assumes independence across dimensions, and that each component is uniformly distributed. This is an unrealistic assumption for real data-sets that typically exhibit significant correlations across dimensions and non-uniform distributions. To approach optimality, an indexing technique must take these properties into account. We resort to a Voronoi clustering framework as it can naturally exploit correlations across dimensions (in fact, such clustering algorithms are the method of choice in the design of vector quantizers [8]). Moreover, we show how our clustering procedure can be combined with any other generic clustering method of choice (such as BIRCH [9]) requiring only one additional scan of the data-set. Lastly, we note that the sequential scan is in fact a special case of clustering based index i.e. with only one cluster.

A. A New Cluster Distance Bound

Crucial to the effectiveness of the clustering-based search strategy is efficient bounding of query-cluster distances. This is the mechanism that allows the elimination of irrelevant clusters. Traditionally, this has been performed with bounding spheres and rectangles. However, hyperspheres and hyperrectangles are generally not optimal bounding surfaces for clusters in high dimensional spaces. In fact, this is a phenomenon observed in the SR-tree [10], where the authors have used a combination spheres and rectangles, to outperform indexes using only bounding spheres (like the SS-tree [11]) or bounding rectangles (R*-tree [12]).

The premise herein is that, at high dimensions, considerable improvement in efficiency can be achieved by relaxing restrictions on the regularity of bounding surfaces (i.e., spheres or rectangles). Specifically, by creating Voronoi clusters, with piecewise-linear boundaries, we allow for more general convex polygon structures that are able to efficiently bound the cluster surface. With the construction of Voronoi clusters under the Euclidean distance measure, this is possible. By projection onto these hyperplane boundaries and complementing with the cluster-hyperplane distance, we develop an appropriate lower bound on the distance of a query to a cluster.

B. Adaptability to Weighted Euclidean or Mahalanobis Distances

While the Euclidean distance metric is popular within the multimedia indexing community (see [7], [13], [14], [15]), it is by no means the "correct" distance measure, in that it may be a poor approximation of user perceived similarities. The Mahalanobis distance measure has more degrees of freedom than the Euclidean distance [6] and by proper updation (or *relevance feedback*), has been found to be a much better estimator of user perceptions (see [16], [17], [5] and more recently [6]) . We extend our distance bounding technique to the Mahalanobis distance metric, and note large gains over existing indexes.

C. An Efficient Search Index

The data set is partitioned into multiple Voronoi clusters and for any kNN query, the clusters are ranked in order of the hyperplane bounds and in this way, the irrelevant clusters are filtered out. We note that the sequential scan is a special case of our indexing, if there were only one cluster. An important feature of our search index is that we do not store the hyperplane boundaries (which form the faces of the bounding polygons), but rather generate them dynamically, from the cluster centroids. The only storage apart from the centroids are the *cluster-hyperplane boundary distances* (or the *smallest* cluster-hyperplane distance). Since our bound is relatively tight, our search algorithm is effective in spatial filtering of irrelevant clusters, resulting in significant performance gains. We expand on the results and techniques initially presented in [18], with comparison against several recently proposed indexing techniques.

III. Performance Measure

The common performance metrics for exact nearest neighbor search have been to count page accesses or the response time. However, page accesses may involve both serial disk accesses and random IOs, which have different costs. On the other hand, response time (*seek times* and *latencies*) is tied to the hardware being used and therefore, the performance gain/loss would be platform dependent.

Hence, we propose a new 2-dimensional performance metric, where we count separately the number of sequential accesses S and number of random disk accesses R. We note that given the performance in terms of the proposed metric i.e. the (S,R) pair, it is possible to estimate total number of disk accesses or response times on different hardware models. The number of page accesses is evaluated as $S + R$. If T_{seq} represented the average time required for one sequential IO and T_{rand} for one random IOs, the average response time would be $T_{res} = T_{seq}S + T_{rand}R$.

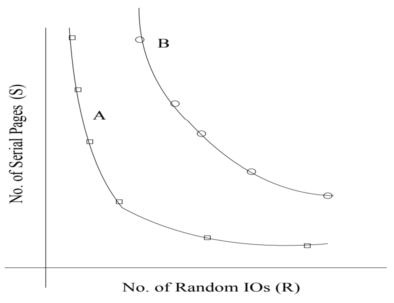

Fig. 1. Comparing Performance Graphs of (hypothetical) Index A and Index B

1) Performance Curves for Index Comparison: Many indexes have index specific *free parameters* which when tuned lead to different performances. For example, in the VA-File the number of quantization levels per dimension can be varied, leading to different performance characteristics. For other indexes like iDistance [15], LDC [19] as well as our own technique , it would be *the number of clusters*. We vary the tunable parameters of each index and obtain a *performance graph*.

If the performance graph of indexing method A ***lies strictly below*** that of another index B (see Figure 1), then method A would be preferable. This is because for every choice of an operating point (S_B,R_B) for method B (or an equivalent response time), we can find an operating point for A (S_A,R_A) that has a smaller R and smaller S and hence, lower response

IV. Related work

Several index structures exist that facilitate search and retrieval of multi-dimensional data. In low dimensional spaces, *recursive* partitioning of the space with hyper-rectangles (R-trees [20], R*-trees [12]), hyper-spheres (SS-Tree [11]) or a combination of hyper-spheres and hyper-rectangles (SR-Tree [10]), have been found to be effective for nearest neighbor search and retrieval. While the preceding methods specialize to Euclidean distance (l_2 norm), M-trees [21] have been found to be effective for metric spaces with arbitrary distance functions (which are metrics).

Such multi-dimensional indexes work well in low dimensional spaces, where they outperform sequential scan. But it has been observed that the performance degrades with increase in feature dimensions and, after a certain dimension threshold, becomes inferior to sequential scan. In a celebrated result, Weber et. al. [7] have shown that whenever the dimensionality is above 10, these methods are outperformed by simple sequential scan. Such performance degradation is attributed to Bellman's '*curse of dimensionality*' [22], which refers to the exponential growth of hyper-volume with dimensionality of the space.

A. Vector Approximation Files

A popular and effective technique to overcome the curse of dimensionality is the vector approximation file (VA-File) [7]. VA-File partitions the space into hyper-rectangular cells, to obtained a quantized approximation for the data that reside inside the cells. Non-empty cell locations are encoded into bit strings and stored in a separate *approximation file*, on the hard-disk. During a nearest neighbor search, the vector approximation file is sequentially scanned and upper and lower bounds on the distance from the query vector to each cell are estimated. The bounds are used to prune irrelevant cells. The final set of candidate vectors are then read from the hard-disk and the exact nearest neighbors are determined. At this point, we note that the terminology "Vector Approximation" is somewhat confusing, since what is actually being performed is *scalar quantization*, where each component of the feature vector is *separately and uniformly quantized* (in contradistinction with vector quantization in the signal compression literature).

VA-File was followed by several more recent techniques to overcome the curse of dimensionality. In the VA$^+$-File [23], the data-set is rotated into a set of uncorrelated dimensions, with more approximation bits being provided for dimensions with higher variance. The approximation cells are adaptively spaced according to the data distribution. Methods such as LDR [24] and the recently proposed non-linear approximations [25] aim to outperform sequential scan by a combination of clustering and dimensionality reduction.There also exist a few hybrid methods, such as the A-Tree [13], and IQ-Tree [14], which combine VA-style approximations within a tree based index.

B. Transformation to and Indexing in One Dimension

Other techniques, such as LDC [19], iDistance [15], and Pyramid Tree [26], are based on local dimensionality reducing transformations. The data-set is partitioned and, in each partition, the distances of the resident vectors to some reference point, typically the centroid, are evaluated. The feature vectors in a partition are now indexed by their centroid-distance, using ubiquitous one-dimensional indexes such as the B$^+$-tree [1]. During query processing, spheres of gradually increasing radii are drawn around the query, until they intersect a cluster sphere. Now, the *relevant* elements in the partition, identified by centroid-distances which lie in the intersecting region, are retrieved for finer scrutiny. The search radius is set to such a value that the exact NNs are returned.

It is to be noted that co-ordinate hyperplanes translated to the centroid divide the feature space into 2^d boxes, where d is the space dimensionality. In LDC [19], another approximation layer is created, by generating a box identification code for each resident point. Once an initial set of candidates have been identified with the B$^+$-tree, the corresponding approximations are scanned to further filter out irrelevant points within the partition. The surviving elements are finally retrieved from the hard drive to determine the nearest neighbors. In order to reduce disk IO, care is taken to control the maximal fraction of space searched, yet return the exact nearest neighbors.

C. Approximate Similarity Search

Lastly, it has been argued that the feature vectors and distance functions are often only approximations of user perception of similarity. Hence, even the results of an exact similarity search is inevitably *perceptually* approximate, with additional rounds of query refinement necessary. Conversely, by performing an approximate search, for a small penalty in accuracy, considerable savings in query processing time would be possible. Examples of such search strategies are MMDR [27], probabilistic searches (PAC-NN) [28], VA-LOW [29], [30], [31], [32] and locality sensitive hashing [33]. The reader is directed to [34] for a more detailed survey of approximate similarity search. The limits of approximate indexing i.e. the optimal tradeoffs between search quality and search time has also been studied within an information theoretic framework [35].

V. Cluster Distance Bounding

In this section, we describe our procedure for estimating distances to clusters. We begin with a few preliminaries and then develop an effective cluster distance bound. A list of notations we use subsequently is presented in Table 1.

Let $d(\mathbf{x}_1, \mathbf{x}_2)$ be a distance function that estimates the distance between *feature vectors* $\mathbf{x}_1$ and $\mathbf{x}_2$, that abstract the actual objects of the database.

$$d : \mathbb{R}^n \times \mathbb{R}^n \rightarrow [0, \infty) \tag{1}$$

[1]Alternatively, consider the tangent to the performance curve of index A with parametric form $T_{seq}S + T_{rand}R = T$, for some specific hardware serial and random IO response times T_{seq}, T_{rand}. The distance of this tangent from the origin is proportional to the optimal performance time with A. Now, if the performance graph of A lies strictly below that of another index B, the tangent to A is closer to the origin than the tangent to B. Hence, A offers a faster response time than B.

TABLE I
TABLE OF NOTATIONS

Symbol	Represents...
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^d$	Feature space of dimension d
$\mathcal{X} \subseteq \mathbb{R}^d$	d-dimensional data-set
$\mathbf{x}, \mathbf{x}_1, \mathbf{x}_2, \ldots$	Typical elements of the data-set
$\mathbf{q}$	Typical query
$\mathcal{X}_1, \ldots, \mathcal{X}_K$	K clusters of the data-set
$\mathbf{c}_m$	Centroid of cluster $\mathcal{X}_m$
$H = (\mathbf{n}, p)$	A hyperplane in the feature space with the normal vector $\mathbf{n}$ and scalar p
$d(\mathbf{q}, H)$	The distance from $\mathbf{q}$ to H
H_{mn}	Hyperplane separating $\mathcal{X}_m$, $\mathcal{X}_n$
$d(.,.)$	A typical distance measure
$d(\mathbf{q}, \mathcal{X}_m)$	The query-cluster distance
$d_{LB}(\mathbf{q}, \mathcal{X}_m)$	A lower bound on query-cluster distance
$d(\mathcal{X}_m, H)$	Distance of cluster $\mathcal{X}_m$ to hyperplane H

In subsequent discussion, we shall first specialize to the Euclidean distance over (real vector spaces) as the feature similarity measure i.e. $d(\mathbf{x}_1, \mathbf{x}_2) = \|\mathbf{x}_1 - \mathbf{x}_2\|_2$ and in a similar fashion, extend results to the Mahalanobis distance measure i.e. $d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T \Sigma^{-1} (\mathbf{x}_1 - \mathbf{x}_2)}$, where Σ is a positive definite matrix.

We define the distance from query $\mathbf{q}$ and a cluster $\mathcal{X}_m$ as

$$d(\mathbf{q}, \mathcal{X}_m) = \min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{q}, \mathbf{x}) \tag{2}$$

We note that it would be necessary to read the elements of the cluster to evaluate $d(\mathbf{q}, \mathcal{X}_m)$. In practice, lower bounds on the query-cluster distance would be suitable approximations. Let $d_{LB}(\mathbf{q}, \mathcal{X}_m)$ be a typical lower bound. Then, by definition,

$$d_{LB}(\mathbf{q}, \mathcal{X}_m) \leq d(\mathbf{q}, \mathcal{X}_m) \tag{3}$$

A. The Hyperplane Bound

Definition 5.1: Hyperplane $H = H(\mathbf{n}, p)$ in $\mathbb{R}^d$ is defined as $\mathcal{H} = \{\mathbf{y} : \mathbf{y}^T \mathbf{n} + p = 0\}$

Definition 5.2: Let $H = H(\mathbf{n}, p)$ be a hyperplane in $\mathbb{R}^d$ and $\mathbf{y} \in \mathbb{R}^d$. Then, the distance of $\mathbf{y}$ from H is $d(\mathbf{y}, H) = \frac{|\mathbf{y}^T \mathbf{n} + p|}{\|\mathbf{n}\|_2}$

We are now ready to develop the hyperplane bound.

Lemma 5.1: Given a cluster $\mathcal{X}_m$, query $\mathbf{q}$ and hyperplane H that lies between the cluster and the query (a "*separating hyperplane*"),

$$d(\mathbf{q}, \mathcal{X}_m) \geq d(\mathbf{q}, H) + \min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{x}, H)$$

Proof: By simple geometry (see Figure 2), it is easy to show that for any $\mathbf{x} \in \mathcal{X}_m$

$$\begin{aligned} d(\mathbf{q}, \mathbf{x}) &\geq d(\mathbf{q}, H) + d(\mathbf{x}, H) \\ &\geq d(\mathbf{q}, H) + \min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{x}, H) \end{aligned}$$

$$\Rightarrow d(\mathbf{q}, \mathcal{X}_m) = \min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{q}, \mathbf{x}) \geq d(\mathbf{q}, H) + \min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{x}, H) \tag{4}$$

■

For convenience, we also use $d(\mathcal{X}_m, H)$ to denote the cluster-hyperplane distance $\min_{\mathbf{x} \in \mathcal{X}_m} d(\mathbf{x}, H)$.

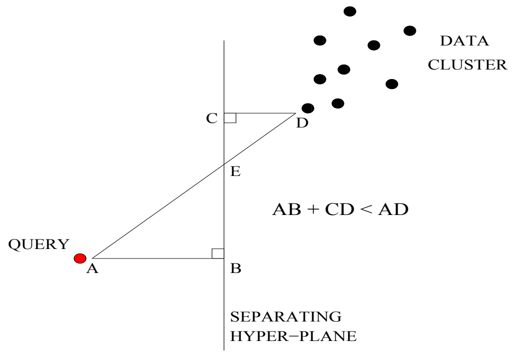

Fig. 2. Principle of the Hyperplane Bound.

Corollary 5.2: If $\mathcal{H}_{sep}$ represents a countably finite set of separating hyperplanes (that lie between query $\mathbf{q}$ and cluster $\mathcal{X}_m$),

$$\Rightarrow d(\mathbf{q}, \mathcal{X}_m) \geq \max_{H \in \mathcal{H}_{sep}} \{d(\mathbf{q}, H) + d(\mathcal{X}_m, H)\} \tag{5}$$

Inequality (5) can be used to obtain a tighter lower bound on $d(\mathbf{q}, \mathcal{X}_m)$. Next, we note that cluster boundaries of Voronoi clusters are indeed piecewise-linear and may hence benefit from the above lower bound (see Figure 3). (This result is of course trivial and is included here for completeness).

Definition 5.3: Let the data-set $\mathcal{X}$ be partitioned into Voronoi clusters $\{\mathcal{X}_m\}$ pivoted around $\{\mathbf{c}_m\}$. Then,

- $\mathcal{X}_m = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}, \mathbf{c}_m) \leq d(\mathbf{x}, \mathbf{c}_n), \forall n \neq m\}$
- $\mathcal{X} = \bigcup_m \mathcal{X}_m$ where $\mathcal{X}_m \bigcap \mathcal{X}_n = \phi, \forall n \neq m$

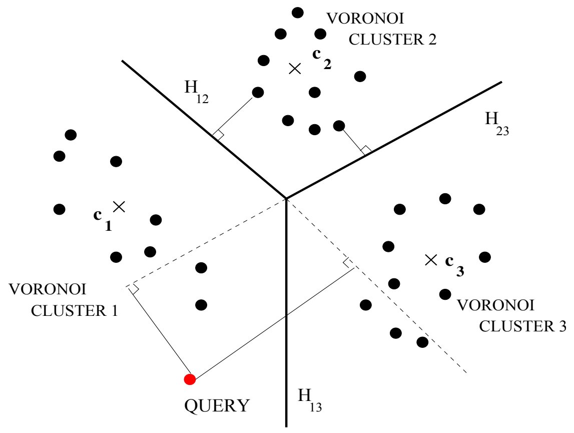

Fig. 3. Query Projection on Separating Hyperplane Boundaries of Voronoi Clusters.

Lemma 5.3: The boundary H_{12} between two Voronoi clusters $\mathcal{X}_1$ and $\mathcal{X}_2$ is a linear hyperplane.

Proof: Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the pivots for $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively. Then, $\forall \mathbf{y} \in H_{12}$

$$\begin{aligned} d(\mathbf{c}_1, \mathbf{y}) &= d(\mathbf{c}_2, \mathbf{y}) \\ \Rightarrow \|\mathbf{c}_1 - \mathbf{y}\|_2^2 &= \|\mathbf{c}_2 - \mathbf{y}\|_2^2 \\ \Rightarrow \|\mathbf{c}_1\|_2^2 - \|\mathbf{c}_2\|_2^2 - 2(\mathbf{c}_1 - \mathbf{c}_2)^T \mathbf{y} &= 0 \end{aligned}$$

Therefore, the hyperplane $H_{12} = H(-2(\mathbf{c}_1 - \mathbf{c}_2), \|\mathbf{c}_1\|_2^2 - \|\mathbf{c}_2\|_2^2)$ is the boundary between the clusters $\mathcal{X}_1$ and $\mathcal{X}_2$. ■

What is to be noted is that these hyperplane boundaries *need not be stored*, rather they can be *generated during run-time* from just the pivots $\{\mathbf{c}_m\}_1^K$. We finally present the condition for a hyperplane to be a separating hyperplane i.e., to lie between a cluster and the query point.

Corollary 5.4: Given a query $\mathbf{q}$ and H_{mn}, the hyperplane boundary between Voronoi clusters $\mathcal{X}_m$ and $\mathcal{X}_n$, H_{mn} lies between $\mathbf{q}$ and cluster $\mathcal{X}_m$ ***if and only if*** $d(\mathbf{q}, \mathbf{c}_m) \geq d(\mathbf{q}, \mathbf{c}_n)$. The proof is fairly straight forward.

Lastly, we show how the boundary between Voronoi clusters remains a hyperplane even under Mahalanobis distances.

Lemma 5.5: The boundary H_{12} between two Voronoi clusters $\mathcal{X}_1$ and $\mathcal{X}_2$, defined by a Mahalanobis distance measure, is a hyperplane.

Proof: Let the new distance measure be evaluated as

$$d(\mathbf{x}_1, \mathbf{x}_2) = \sqrt{(\mathbf{x}_1 - \mathbf{x}_2)^T \Sigma^{-1} (\mathbf{x}_1 - \mathbf{x}_2)} \qquad (6)$$

Let $\mathbf{c}_1$ and $\mathbf{c}_2$ be the pivots for $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively.

$$\begin{aligned} \forall \mathbf{y} \in H_{12}, d(\mathbf{c}_1, \mathbf{y}) &= d(\mathbf{c}_2, \mathbf{y}) \\ \Rightarrow (\mathbf{c}_1 - \mathbf{y})^T \Sigma^{-1} (\mathbf{c}_1 - \mathbf{y}) &= (\mathbf{c}_2 - \mathbf{y})^T \Sigma^{-1} (\mathbf{c}_2 - \mathbf{y}) \\ \Rightarrow -2(\mathbf{c}_1 - \mathbf{c}_2)^T \Sigma^{-1} \mathbf{y} &+ \mathbf{c}_1^T \Sigma^{-1} \mathbf{c}_1 - \mathbf{c}_2^T \Sigma^{-1} \mathbf{c}_2 = 0 \end{aligned}$$

Therefore, H_{12}, the boundary between the clusters $\mathcal{X}_1$ and $\mathcal{X}_2$, is a hyperplane. ∎

B. Reduced Complexity Hyperplane Bound

For evaluation of the lower-bound presented in (4) and (5), we would need to pre-calculate and store $d(H_{mn}, \mathcal{X}_m)$ for all cluster pairs (m, n). Hence there are $K(K-1)$ distances that need to be pre-calculated and stored, in addition to the cluster centroids themselves. The total storage for all clusters would be $O(K^2 + Kd)$. This storage can be reduced by further loosening the bound in (5) as follows:

$$\begin{aligned} d(\mathbf{q}, \mathcal{X}_m) &\geq \max_{H \in \mathcal{H}_{sep}} \{d(\mathbf{q}, H) + d(H, \mathcal{X}_m)\} \\ &\geq \max_{H \in \mathcal{H}_{sep}} d(\mathbf{q}, H) + \min_{H \in \mathcal{H}_{sep}} d(H, \mathcal{X}_m) \end{aligned}$$

This means that for every cluster $\mathcal{X}_m$ we would only need to store one distance term

$$d_m = \min_{1 \leq n \leq K, n \neq m} d(H_{mn}, \mathcal{X}_m)$$

VI. Clustering and Index Structure

The first step in index construction is the creation of Nearest Neighbor/Voronoi clusters. There exist several techniques of clustering the data-set, from the fast K-means algorithm [36] (which requires multiple scans of the data-set) and Generalized Lloyd Algorithm (GLA) [8] to methods such as BIRCH [9], which require only a single scan of the data-set. The output of any of these algorithms can be a starting point. From each of the K clusters detected by a generic clustering algorithm, a pivot is chosen i.e. K pivot points in all. Then the entire data-set is scanned and each data-element is mapped to the nearest pivot.

Data mapping to the same pivot are grouped together to form Voronoi clusters (see Algorithm 1). This would lead to slight re-arrangement of clusters, but this is necessary to retain piecewise linear hyperplane boundaries between clusters. We believe the centroid is a good choice as a pivot. Thus, quick Voronoi clustering, with possibly only a single scan of the entire data-set, can be achieved using any generic clustering algorithm. Lastly, also note that any indexing scheme would need at least one scan of the database, which indicates that index construction times for our scheme are as very close to the minimum possible.

Algorithm 1 VORONOI-CLUSTERS($\mathcal{X}$,K)

```
1: //Generic clustering algorithm returns
   //K cluster centroids
   $\{\mathbf{c}_m\}_{=1}^K \leftarrow$GenericCluster($\mathcal{X}$,$K$)
2: set $l = 0$, $\mathcal{X}_1 = \phi, \mathcal{X}_2 = \phi, \ldots, \mathcal{X}_K = \phi$
3: while $l < |\mathcal{X}|$ do
4:     $l = l + 1$
5:     //Find the centroid nearest to data element $\mathbf{x}_l$
       $k = \arg\min_m d(\mathbf{x}_l, \mathbf{c}_m)$
6:     //Move $\mathbf{x}_l$ to the corresponding Voronoi partition
       $\mathcal{X}_k = \mathcal{X}_k \bigcup \{\mathbf{x}_l\}$
7: end while
8: return $\{\mathcal{X}_m\}_{=1}^K, \{\mathbf{c}_m\}_{=1}^K$
```

We note that the K-means, GLA and BIRCH algorithms are fast and can generate reliable estimates of cluster centroids, from sub-samples of the data-set. Typically, for K clusters, even a sub-sample of size $100K$ is sufficient. As we shall see, for the range of clusters we are considering, this would be overwhelmingly smaller than the data-set. Faster index construction would be possible by allowing for hierarchical and multi-stage clustering. However, only the clusters at the leaf level are returned.

We tested several clustering techniques including GLA and BIRCH, and the results were largely similar. While it is possible to also optimize the clustering itself, that is not our goal in these experiments.

A. Storage Strategy

Elements within the same cluster are stored together (contiguously). We retain the cluster centroids $\mathbf{c}_m$ and maintain pointers from each centroid to the location of the corresponding cluster on the hard-disk. We also maintain in a separate file the distance (bounds) of each cluster from its bounding hyperplanes. We note that the total storage is $O(Kd + K^2)$ and $O(K(d+1))$ real numbers, for the full and reduced complexity hyperplane bounds respectively, where K is the number of clusters. Lastly, note that while the elements of each cluster are stored contiguously, the individual clusters are stored on different parts of the disk, with enough space provided to allow them to grow and shrink.

Again, we note that the hyperplane boundaries can be generated in run-time and assume systems of sufficient main-memory capacity to allow storage of intermediate results. For a main memory capacity of G Gigabytes, and assuming the exact hyperplane bound with $K >> d$, we can store $O(10^4 \times \sqrt{G})$ clusters. Typical database server memory would easily handle

TABLE II
ADDITIONAL NOTATION

Variable	Type	Represents
N	Output	No. of elements read
$d_{LB}^{sort}[...]$	Internal	Query-cluster lower bounds sorted in ascending order
$o[...]$	Internal	Ranking of clusters (by distance bounds)
$kNN[...]$	Output	k-nearest neighbors to **q**
d_{kNN}	Internal	Distance of the k^{th}-NN found so far
$\mathcal{X}_{cand}$	Internal	Candidate set
$VecsPerPage$	Internal	No. of d-dim. vectors that fit on a page
FLAG	Internal	Flag to continue search or stop
$count$	Internal	Internal counter

the range of clusters we consider. Figure 4 is representative of the proposed index.

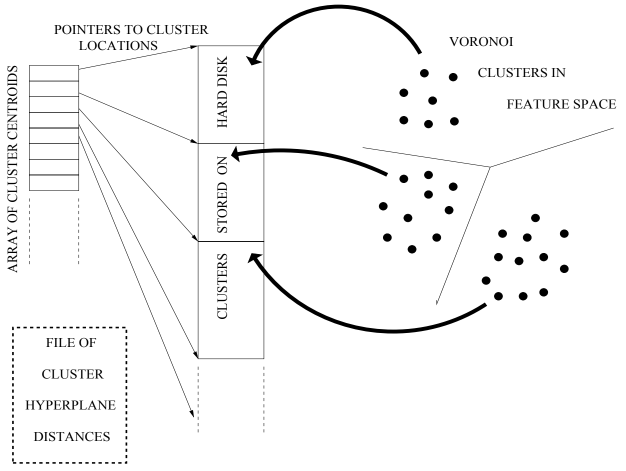

Fig. 4. Proposed Index Structure.

VII. THE KNN SEARCH ALGORITHM

We now present KNN-SEARCH, our procedure for k-NN search. Our algorithm is a branch-and-bound algorithm, and since the clusters are accessed in order of the lower bounds to the query distance, it is *guaranteed* to return the k-nearest neighbors. The main search algorithm KNN-SEARCH (see Algorithm 2) calls on four functions

- HyperplaneBound(**q**) - returns lower bounds on the distance between query **q** and all clusters, using separating hyperplane boundaries (as explained in section V-A).
- SortArray(a[],'ascend') - sorts an array a in ascending order and returns the sorted array and sorting "rank" (order).
- Farthest($\mathbf{x}, \mathcal{A}$) - returns the distance of the element in $\mathcal{A}$ furthest from **x**.
- FindkNNsIn($\mathbf{q}, \mathcal{A}, \mathcal{I}$) - for query **q** and initial candidate list $\mathcal{I}$, finds and returns the kNNs in cluster $\mathcal{A}$, as well the number of elements in the cluster
- LoadNextPage(F) - loads a page of a cluster into main memory (RAM), where F is the pointer to the cluster file
- Find$_{kNN}(\mathbf{x}, \mathcal{A})$ - finds the k-nearest neighbors of **x** in set $\mathcal{A}$.

Algorithm 2 KNN-SEARCH(**q**)

```
1: //Initialize
   set FLAG=0, $count = 0, N = 0, kNN = \phi$
2: //Evaluate query-cluster distance bounds
   $d_{LB}[]$ ←HyperplaneBound(q)
3: //Sort the query-cluster distance bounds in ascending
   //order
   $\{d_{LB}^{sort}[], o[]\}$ ←SortArray($d_{LB}$,'ascend')
4: while FLAG==0 do
5:    $count = count + 1$
6:    //Find the kNNs upto current cluster
      $\{N_c, kNN\}$ ←FindkNNsIn(q, $\mathcal{X}_{o[count]}, kNN$)
7:    //Update number of elements scanned
      $N = N + N_c$
8:    //Find the kNN radius
      $d_{kNN}$=Farthest(q, $kNN$)
9:    if $count < K$ then
10:      if $N > k$ then
11:         if $d_{kNN} < d_{LB}^{sort}[count + 1]$ then
12:            set FLAG=1 //kNNs found, search ends
13:         end if
14:      end if
15:   else
16:      set FLAG=1 //all clusters scanned, search ends
17:   end if
18: end while
19: return $kNN$
```

Algorithm 3 FindkNNsIn($\mathbf{q}, \mathcal{A}, \mathcal{I}$)

```
1: set $N_c = 0$, $F$=Open($\mathcal{A}$), $kNN = \mathcal{I}$
2: while !(EOF($F$)) do
3:    // Load the next cluster page
      $\mathcal{C}$=LoadNextPage(F)
4:    //Merge kNN list with current page
      $\mathcal{X}_{cand} = \mathcal{C} \bigcup kNN$
5:    //Find the kNNs within the candidate list
      $kNN[]$ ←Find$_{kNN}$(q, $\mathcal{X}_{cand}$)
6:    //Update number of elements scanned
      $N_c = N_c + |\mathcal{C}|$
7: end while
8: return $N_c, kNN$
```

For every query, the processing starts with a call to HyperplaneBound(**q**). The centroids and the cluster-hyperplane distances are loaded into the main memory. The hyperplane bounds are calculated and returned. These lower bounds are sorted in ascending order and the clusters are correspondingly ranked (line 3). Then, the first (nearest) cluster is scanned from the hard-disk (one page at a time[2], see Algorithm 3) and the kNNs within this subset of the data-set (line 6)

[2]We scan clusters one page at a time, since cluster sizes might exceed available main memory (RAM). As a result, the kNN candidate evaluation is pipelined or threaded with cluster scanning

are identified. Additionally, d_{kNN}, the distance of the k^{th}-NN from the query, is evaluated (line 8) and stored in main memory. If this distance is less than the distance of the next closest cluster, then the search ends as the kNNs have been found. In other words, this is the stopping condition (line 12).

Otherwise, the second cluster is scanned from the disk. The previous estimate of the kNNs is merged with the entries of the current cluster to form a new candidate set and the kNNs within this new candidate set are evaluated. This search procedure continues till the stopping condition is reached or all clusters have been searched (line 16).

VIII. Experimental Results

We have conducted extensive tests on real data-sets to characterize and compare the performance of our index structure with other popular schemes. By varying the number of clusters (or some suitable parameter), a range of solutions, in terms of random vs. sequential IOs, was obtained for each index and these performance curves were subsequently compared.

A. Data-sets and Experimental Set-up

We tested our index on 5 different data-sets - HISTOGRAM, SENSORS, AERIAL, BIO-RETINA and CORTINA. The HISTOGRAM[3] data-set consisted of color histogram extracted from a sample image data-set. The second real data-set, SENSORS, was generated by the Intel Berkeley Research Lab[4]. Data were collected from 54 sensors deployed in the Intel Berkeley Research lab between February 28 and April 5, 2004. Each sensor measures humidity, temperature, light and voltage values once every 31 seconds. We retain data from those sensors that generated in excess of 50,000 readings. This corresponds to 4 types of measurements - temperature, light, humidity and voltage readings - from 15 sensors which is equivalent to 60 correlated sources.

The next two data-sets, AERIAL and BIO-RETINA, were MPEG-7 texture feature descriptors extracted from 64×64 tiles of the images. AERIAL[5] was extracted from 40 large aerial photographs while BIO-RETINA [6] was generated from images of tissue sections of feline retinas as a part of an ongoing project at the Center for Bio-Image Informatics, UCSB. On the other hand, the CORTINA[7] data-set consists of a combination of homogenous texture features, edge histogram descriptors and dominant color descriptors extracted from images crawled from the World Wide Web. In our experiments, we assumed a *page size* of 8kB. The query sets were 100 randomly chosen elements of the relevant data-sets. In all subsequent sections, we report results from experiments where the 10 nearest neighbors (10NN) were mined, unless otherwise stated (see section VIII-I, where the number of kNNs retrieved was varied).

[3]Download from http:/scl.ece.ucsb.edu/datasets/Histogram.mat
[4]Download from http://db.csail.mit.edu/labdata/labdata.html
[5]Download from http://vision.ece.ucsb.edu/data-sets
[6]Download from http://scl.ece.ucsb.edu/datasets/BIORETINA_features.txt
[7]http://scl.ece.ucsb.edu/datasets/CORTINA_all_feat1089K.mat

TABLE III
Data-Sets Used

Name	Dimensionality	No. of Vectors	Size (Pages)
HISTOGRAM	64	12,103	379
SENSORS	60	50,000	1471
AERIAL	60	275,465	8300
BIO-RETINA	62	208,506	6200
CORTINA	74	1,088,864	40,329

TABLE IV
Tunable Parameter Variation in Experiments

Indexing Method	Tunable parameter	Range Tested
VA-File	Bits per dimension	3 - 12 bits
VA$^+$-File	Bits per dimension	5 - 8 bits
iDistance	No. of clusters	10 - 400
LDC	No. of clusters	10 - 4000
Clustering + Cluster-distance bounding	No. of clusters	10 - 400

B. Performance Measure and Tunable Parameter Variation

We propose to *separately* track the *average* number of sequential and random accesses incurred to retrieve the 10NNs. In a typical search procedure, by varying the tunable parameters θ (where applicable), a range of solutions with different sequential and random accesses would be possible i.e. a *performance graph*. Two competing search procedures are compared by comparing the number of random seeks R given roughly the same S or vice-versa.

Table IV lists the tunable parameter for each indexing technique and the range of values considered. The performance of each indexing technique is evaluated at all values of its tunable parameter within the ranges considered. In the VA-File, the number of quantizing bits is varied from 3 to 12 bits per dimension, while in the VA$^+$-File, this is varied from 5-8 bits per dimension. In LDC, the number of clusters is varied from 10 - 4000 clusters, whereas in the iDistance and our and our proposed technique, the number of clusters was varied from 10-400. The performance is evaluated and plotted at each setting.

C. Comparison with Rectangular and Spherical Distance Bounds

Traditionally, spatial filtering of irrelevant clusters has been done by constructing minimum bounding hyperrectangles (MBR) and hyperspheres (MBS) around clusters, and evaluating distance bounds to these minimum bounding surfaces (see Figure 5). We, on the other hand, evaluate this distance bound from the separating hyperplane boundaries that lie between the query and the cluster.

In Figures 6 and 7, we present MBR, MBS and Hyperplane distance bounds along with the true (golden) query-cluster distances for a sample query, on the BIO-RETINA data-set. The distance of the query to every cluster is evaluated and these distances are sorted in ascending order. We note that his sorting order or ranking is also the order in which the clusters are read. Now, the distances bounds are evaluated to these *ranked* clusters and are compared with the true distances. The goal of this study is to compare the relative tightness of the

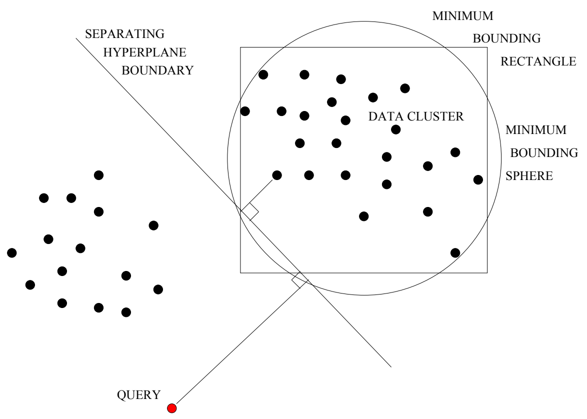

Fig. 5. Separating Hyperplane vs. Minimum Bounding Hypersphere and Rectangle

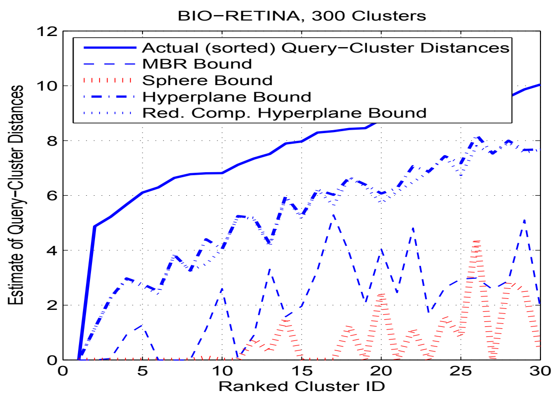

Fig. 7. Comparison of Distance Bounds, 300 Clusters

bounds and also to observe how well they imitate the correct ranking.

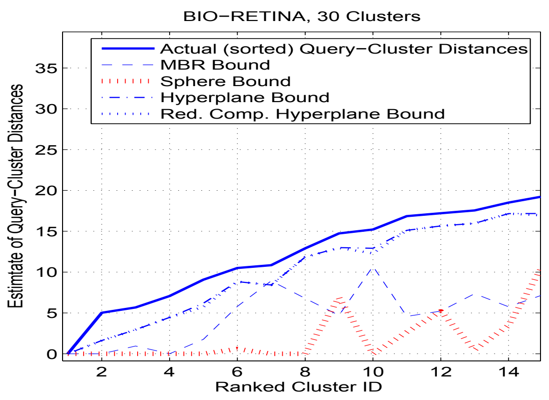

Fig. 6. Comparison of Distance Bounds, 30 Clusters

We immediately note that the hyperplane bounds are very close to the true distance. For the 30 cluster case, we note that the sphere (MBS) bound is *almost zero* for the first 6-8 clusters. This is because in a large cluster (small number of clusters), the volume occupied by bounding spheres is significantly larger than the actual cluster volume. Hence, the ability to filter out clusters is significantly diminished. Once the number of clusters has been increased to around 300, the performance of the sphere bound improves slightly. On the other hand, though the MBR bound is tighter than the sphere bound, it is much looser than the hyperplane bounds. This suggests that while the MBR is able to adjust to the cluster shape a little better than the MBS, the convex polygons formed by the hyperplane boundaries adapt to the cluster shape much better than both MBRs and MBSs. We noticed similar behavior in the other data-sets.

Lastly, we note that, the reduced complexity hyperplane bound is almost as good as the full complexity hyperplane bound, while at the same time enjoying a smaller storage overhead. We also note that the *distance profiles* generated by the hyperplane bounds has almost the same (non-decreasing) nature of the true, sorted query-cluster distances. This means that despite the approximation error in distance estimates, *the clusters are still searched in close to the correct order*. However, this is not so for the MBR and MBS bounds. We note in Figures 6 for the MBR bound and 7 for the MBS bound, the distance profile is very jittery. This, as we shall see, leads to a large number of needless disk accesses, further compromising the IO performance.

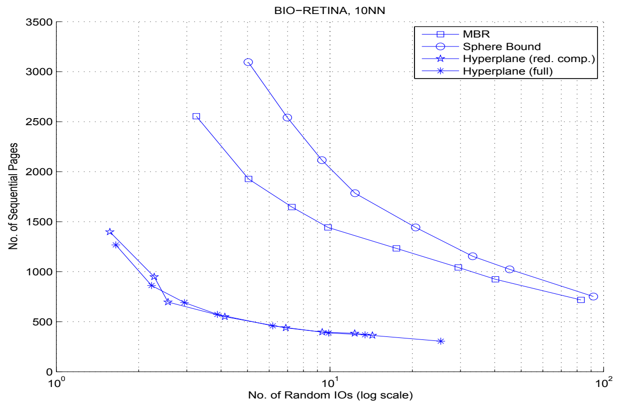

Fig. 8. IO Performance of Distance Bounds - BIO-RETINA

1) IO Performance Comparison: We conclude this study with comparison of the IO performance of the different distance bounds on the different data-sets (see Figures 8, 9, 10, 11, 12). We note that the sphere (MBS) and the MBR bounds are comprehensively outperformed by the two proposed hyperplane bounds in all data-sets. For the BIO-RETINA data set, when allowed roughly 1000 sequential IO pages, the MBR and sphere bounds generate nearly 20X *more* random accesses than the hyperplane bounds. When allowed roughly 10 random disk accesses, the MBR and sphere require nearly 4X-5X more sequential disk reads. Similar trends are noticed in other data-sets, with the performance gains higher in bigger data-sets.

D. Comparison with Popular Indexes

Next, we compare the performance of our proposed clustering (plus 'Hyperplane Bound' or 'Red. Comp. Hyperplane Bound') framework with the VA-File, VA$^+$-File, iDistance and

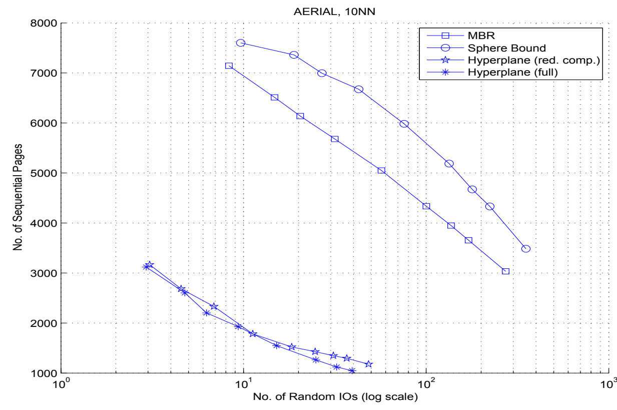

Fig. 9. IO Performance of Distance Bounds - AERIAL

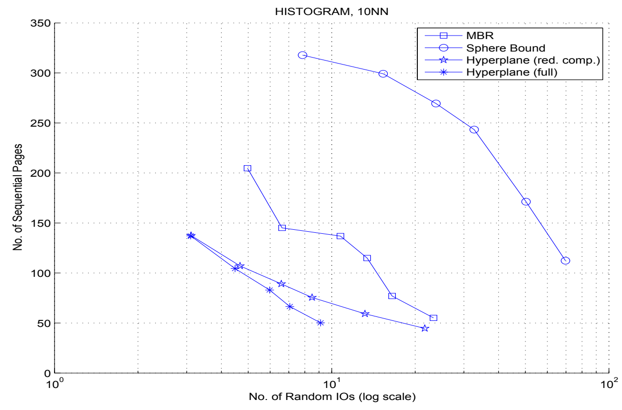

Fig. 11. IO Performance of Distance Bounds - HISTOGRAM

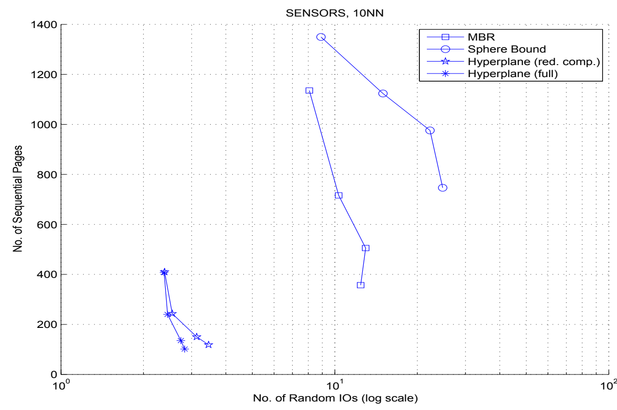

Fig. 10. IO Performance of Distance Bounds - SENSORS

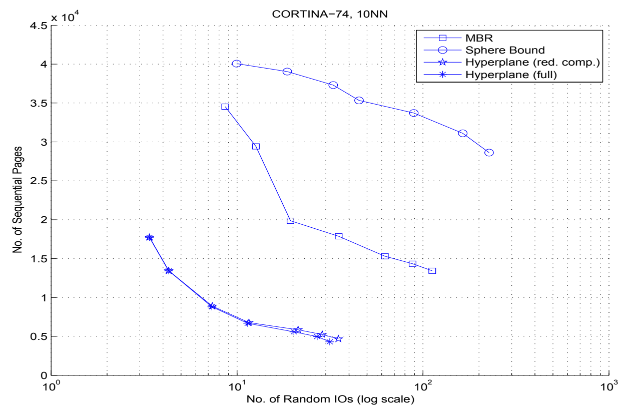

Fig. 12. IO Performance of Distance Bounds - CORTINA

Local Digital Coding (LDC) - recently proposed multidimensional index structures that have been successful in overcoming the curse of dimensionality.

1) Description: In the VA-File/VA$^+$-File initially the approximation files are read into the main memory and then the nearest elements are retrieved for finer scrutiny. As more bits are allocated in quantizing the dimensions, the size of the approximation file, which is proportional to the no. of sequential IOs, increases. At the same time, the number of vectors visited in the second stage, which determines the no. of random IOs, are reduced. Hence, the IO performance was evaluated at different levels of compression (VA-File - 12 to 3 bits/dimension and VA$^+$-File - 8 to 5 bits/dimension).

For the proposed index structure, as well as the iDistance and the LDC, the performance would depend on the number of clusters created. The number of clusters was varied from 10-4000 for LDC and 10-400 for iDistance and our technique. Each multi-dimensional index was searched till the 10 *exact* nearest neighbors (10NNs) were *guaranteed* to be found.

In all data-sets (Figure 13, 14, 15, 16 and 17), the proposed index (clustering plus hyperplane bounds) outperforms all popular indexes. For the CORTINA data-set, when allowed (roughly) the same number of sequential IOs as the VA-File, speed-ups in costly random IOs ranging from 3000X-5X, are possible. For clarity, we also present in Table V the actual random IOs incurred for different sequential IOs in the four best indexing schemes i.e. the VA-File, VA$^+$-File, iDistance and our proposed index with full hyperplane bound. In the AERIAL data-set, when allowed (roughly) the same number of sequential IOs as the VA-File, our index structure has random IOs reduced by factors ranging from 500X-5X. When allowed the same number of sequential page accesses, random disk IOs reductions ranging from 40X-90X over the LDC, 3x-16x over the iDistance and 1.6X over the VA$^+$-File were observed.

2) Discussion: The basis behind the idea of vector approximations is that if a compressed feature vector is close to the query, so is the original feature vector. To generalize the idea behind vector approximations, if multiple feature vectors were represented by a single code-vector, and if this code-vector is close to the query, so would the original feature vectors. By clustering the data-set, we group together feature vectors that are similar and represent the cluster by the centroid. Hence, by accessing the nearest clusters, a bigger candidate set of feature vectors is retrieved.

Moreover, in the second phase of the VA-File/VA$^+$-File search, relevant feature vectors are selected, based upon distance approximations, for retrieval by random access operations. For a k-NN query, this implies that a minimum of k-random IO reads is *inevitable*. More importantly, it is to be noted that in any random access read, even though *an entire page* is read into the main memory, only *one feature vector* is relevant and the remaining data from the page is not useful. For

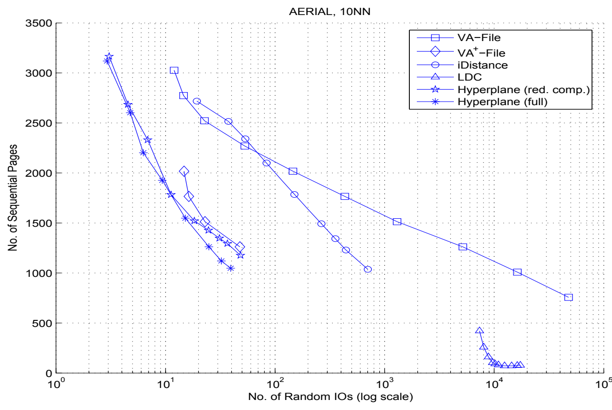

Fig. 13. IO Performance of Indexes - AERIAL

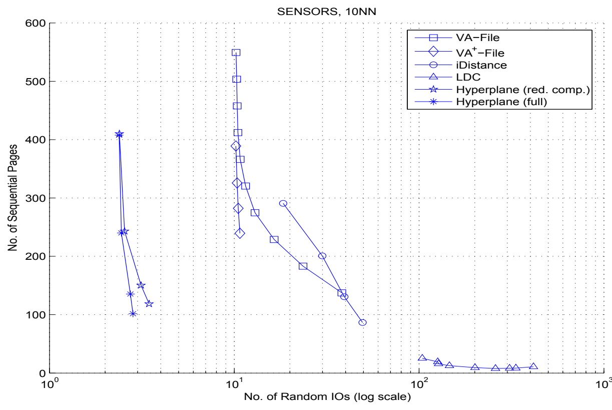

Fig. 15. IO Performance of Indexes - SENSORS

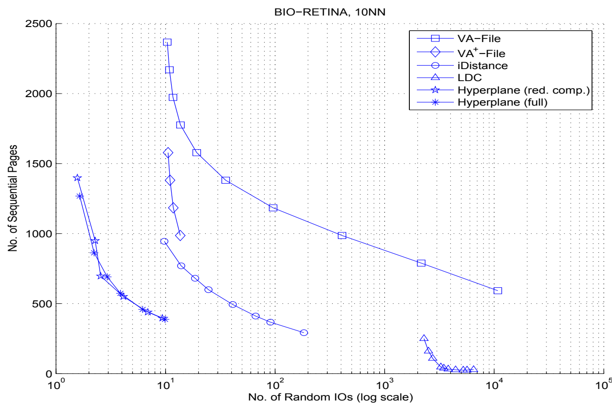

Fig. 14. IO Performance of Indexes - BIORETINA

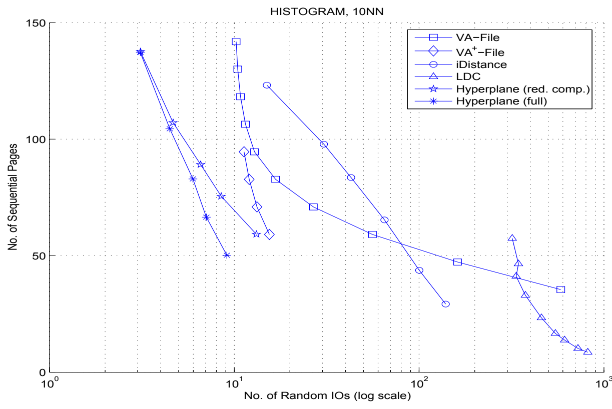

Fig. 16. IO Performance of Indexes - HISTOGRAM

example, with a page size of 8kB and 60-dimensional (floating point vectors), each random access read would retrieve the equivalent of 34 feature vectors in order to obtain a single feature vector. This inefficient use of random IOs is avoided in our clustering scheme where every random IO retrieves the first page of cluster, yielding a bunch of candidate feature vectors.

Lastly, we note that the iDistance and LDC indexes select clusters based on distance bounds to different bounding spheres. As observed in prior sections, this bound is loose and results in poor spatial filtering of the data-set. We also note that LDC suffers from a weakness also present in the VA-File. While pruning the cluster with relevant digital codes (DCs), the final candidate vectors still need to be retrieved by *individual* random disk reads. By maintaining low-precision digital codes, the number of sequential reads are reduced but as a consequence, a fairly large number of expensive random IOs are necessary to retrieve the final candidate vectors.

TABLE V
RANDOM IO COSTS FOR CORTINA 10NN QUERIES

Indexing Scheme \ Serial IO (pages)	6680	8800	13,430
VA-File	11910	290	13.18
VA$^+$-File	15	11.4	10.2
iDistance	322	180	55.16
Cluster Distance Bounding Hyperplane (full)	11.41	7.32	4.29

E. Computational Costs

We also evaluated the computational costs incurred by the different indexes, in terms of the number of distance evaluations. Since the VA-File/VA$^+$-File maintains an approximation for every element of the data-set and evaluates the distances to these approximation cells, the number of distance calculations are $O(|\mathcal{X}|)$, the size of the data-set. However, we note that the other indexes are based on clustering, which can exploit spatial dependencies across dimensions. Initial distances are evaluated to the cluster centroids and *relevant clusters* alone are inspected. This results in efficient filtering and substantial reductions in the number of distance computations required.

We only present results from the CORTINA data-set, as similar trends were observed for other data-sets. Our index structure requires ≈10X less distance calculations than the VA-File or VA$^+$-File on the average. It also compares favorably with the iDistance. In the case of LDC, with its combination of dimension ranking arrays and partial distance searches, explicit distance evaluations for the second layer of approximations, are replaced by faster binary operations. Nevertheless, since processor speeds are much faster than IO speeds, out index structure still maintains its advantage in total response time

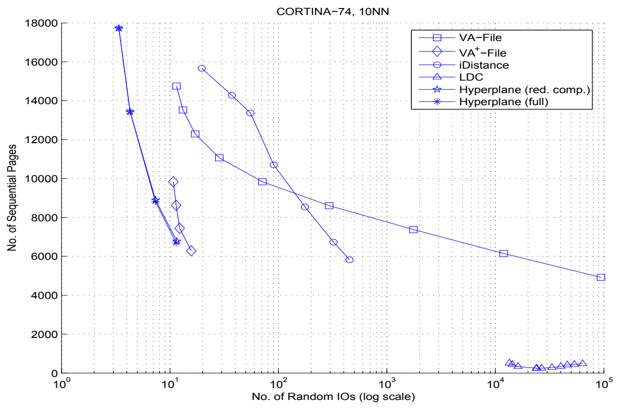

Fig. 17. IO Performance of Indexes - CORTINA

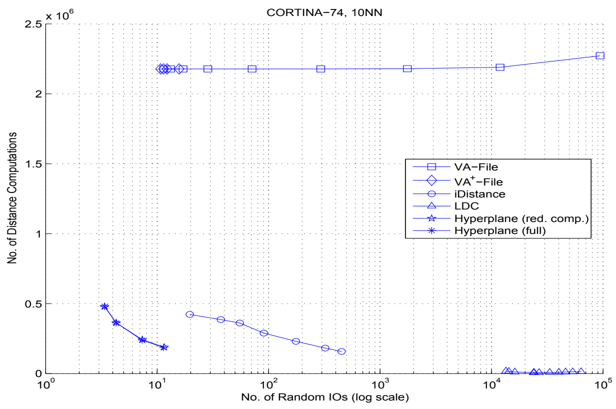

Fig. 18. Computational Costs - CORTINA

over LDC.

F. Preprocessing Storage

In Table V, we evaluate the preprocessing storage for each indexing scheme in terms of the various entities, including dimensionality d, database size N, approximation bits per dimension b, and total number of clusters K. We assume 4 bytes of storage for each real (floating point) number. Note that in the reduced complexity hyperplane bound, we need to store only one distance term per cluster, while in the full complexity hyperplane bound, we have to store $K-1$ distance terms per cluster.

We present the preprocessing storage vs. random IOs performance for the CORTINA data-set and notice substantial gains over competing indexes (Figure 19). This is because on the one hand the approximation file size grows with data-set size, dimensions and the number of approximation/quantization bits per dimension. On the other hand, both iDistance and LDC store one distance term for every data-set element (in addition to the centroids), thus incurring enormous preprocessing storage costs. For the same number of random IOs, our proposed index requires ≈100X less storage than the VA-File or VA$^+$-File.

TABLE VI
PREPROCESSING STORAGE FOR INDEXING SCHEMES

Indexing Scheme	Storage (bytes)
VA-File	$\frac{1}{8}.N.d.b$
VA$^+$-File	$\frac{1}{8}.N.d.b$
iDistance	$4.(K.d+N)$
LDC	$4.(K.d+N)+\frac{1}{8}.N.d$
Hyperplane Bound (full complexity)	$4.K.d+4.K.(K-1)$
Hyperplane Bound (reduced complexity)	$4.K.(d+1)$

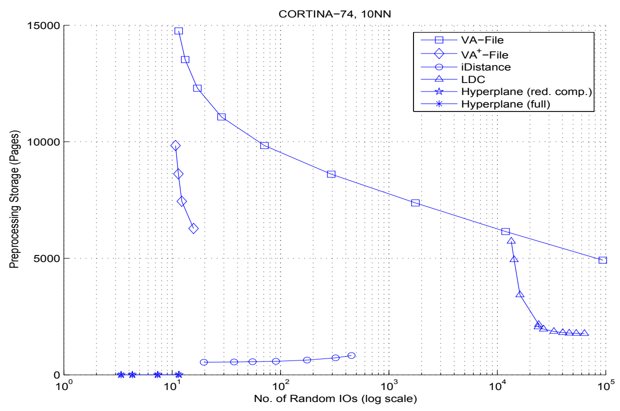

Fig. 19. Preprocessing Storage - CORTINA

G. Scalability with Data-set Size

Figures 20 and 21 present performance comparisons of the proposed indexes with sub-sampled versions of the CORTINA data-set. Figure 20 represents the results of varying the data-set size for the full complexity hyperplane bound. Figure 21 pertains to the reduced complexity hyperplane bound. The performance variation is nearly linear in the number of sequential accesses and almost insensitive in the number of random IOs. Hence, our clustering based index scales well with data-set size.

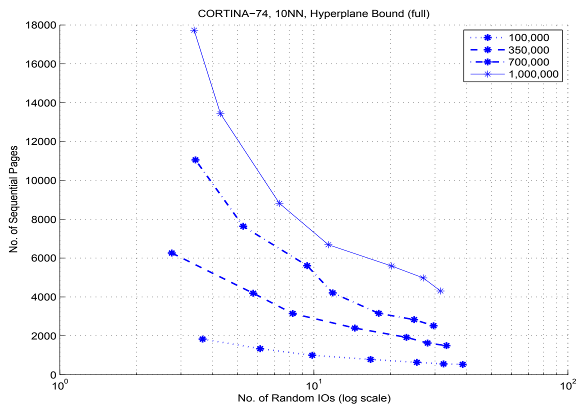

Fig. 20. Data-set Size vs. Full Complexity Hyperplane Bound

H. Scalability with Dimensionality

We also evaluated the scalability of the proposed indexes with dimensions by retaining only 48, 24, 10 and 1 dimensions of the original CORTINA data-set (Figures 22, and 23

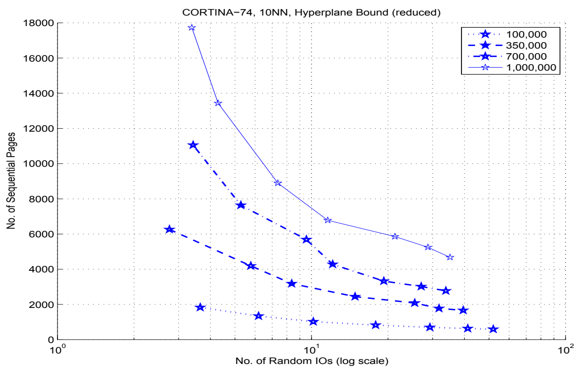

Fig. 21. Data-set Size vs. Reduced Complexity Hyperplane Bound

respectively). Both methods display a graceful degradation in performance with dimensionality. We also note that the VA-File degrades at very low number of dimensions while naive indexes degrade at high dimensionality.

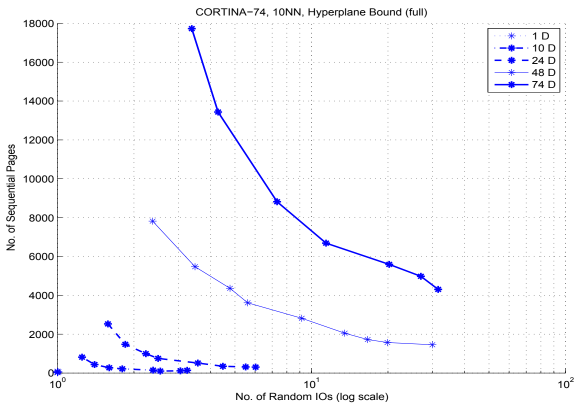

Fig. 22. Dimensionality vs. Full Complexity Hyperplane Bound

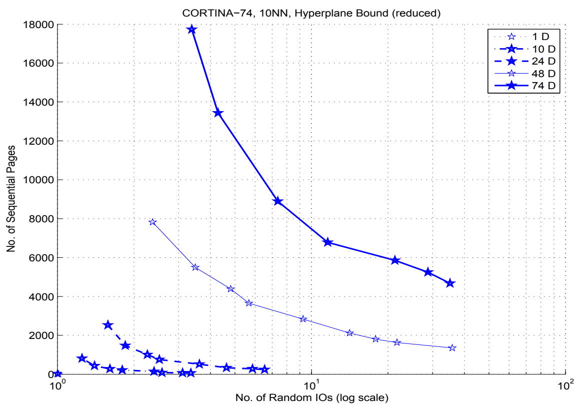

Fig. 23. Dimensionality vs. Reduced Complexity Hyperplane Bound

I. Scalability with Number of Nearest Neighbors

We also vary the number of nearest neighbors with 10NNs, 20NNs and 50NNs being mined (see Figures 24 and 25). We note very slight variation in performance. This is because when each cluster is retrieved, several promising candidates are available. Hence, even if we search for more NNs, we don't notice any substantial increase in disk accesses.

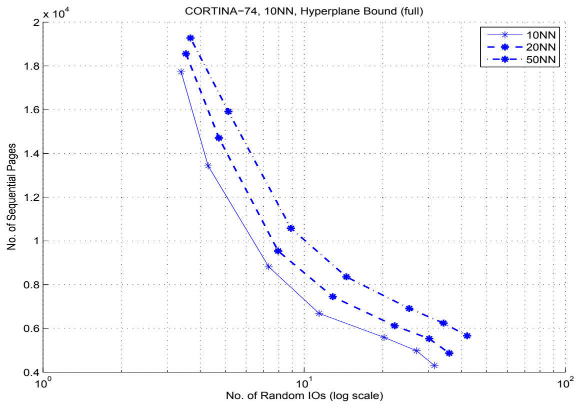

Fig. 24. No. of kNNs vs. Full Complexity Hyperplane Bound

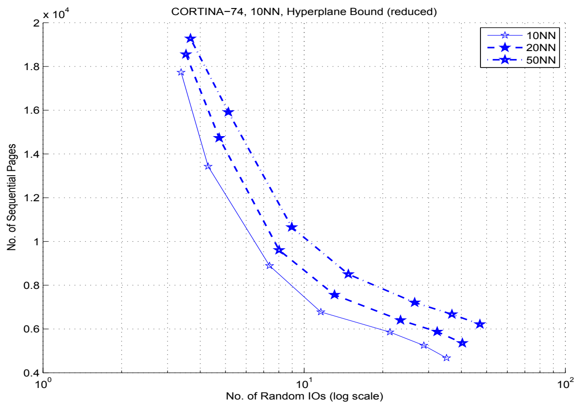

Fig. 25. No. of kNNs vs. Reduced Complexity Hyperplane Bound

J. Robustness to Random Insertions and Deletions

Our index structure is robust to insertions and deletions as may be evident from the design procedure itself. We first note that our index structure is flat or scan-based (and not tree based). Next, while we store elements of a cluster together, the clusters themselves may be located in different parts of the disk i.e. we can provide enough space for clusters to grow/shrink. Further, during index construction, the centroids are extracted using only *random* subsamples of the data-set and are used as pivots to induce Voronoi space partitions. But given a set of centroids, the creation of the index structure is almost complete, since the Voronoi partitions created are *independent* of the data that might reside in them. In other words, during creation, the index is *blind* to the presence or absence of the remaining elements. The creation of the index from random subsamples and subsequent casting of all database elements into Voronoi partitions can be considered equivalent to deletion and insertion, respectively.

In case of an insertion, the only additional complexity involves the possible update of its cluster distance to a hyperplane if the new point is closer than the current bound. In a similar manner, deletions can also be handled, by reevaluating (and if *necessary* updating) the cluster-hyperplane distances. To reduce calculations (at the expense of a negligible storage overhead), an ordered list of cluster-hyperplane distance(s) can be maintained and periodically updated.

In our experiments, for runs with more than 400 clusters, hierarchical clustering was performed. The data-set is initially partitioned into 400 clusters and later these clusters are appropriately sub-clustered. This is because what is necessary for index construction are only the cluster centroids, which act as the pivot points for the subsequent Voronoi partitioning. In either case, with K clusters, only *random* subsamples of the data-set, of size $100K$ vectors, were used in creating the index. For example, under this setting, *only* 40,000 out of 1,088,864 elements of CORTINA were used in index construction.

K. Extension to Approximate Search

Since feature vectors are imperfect representations of the corresponding objects, it could be argued that even exact search is unavoidably approximate. Since each disk IO retrieves a cluster, it may be sufficient to stop the search after the first few disk accesses to extract an approximate result, that could be further refined with user feedback. For brevity of presentation, we describe results for only the reduced complexity hyperplane bound. Similar results are observed with the full complexity hyperplane bound.

In approximate similarity search, the quality of the retrieved is typically measured through precision or recall metrics. If $\mathcal{A}(\mathbf{q})$ and $\mathcal{G}(\mathbf{q})$ represent the approximate and golden (true) answer sets for query $\mathbf{q}$, we define

$$\begin{aligned} Precision &= \frac{|\mathcal{A}(\mathbf{q}) \cap \mathcal{G}(\mathbf{q})|}{|\mathcal{A}(\mathbf{q})|} \\ Recall &= \frac{|\mathcal{A}(\mathbf{q}) \cap \mathcal{G}(\mathbf{q})|}{|\mathcal{G}(\mathbf{q})|} \end{aligned}$$

For kNN queries, $|\mathcal{A}(\mathbf{q})|=|\mathcal{G}(\mathbf{q})|$ and hence precision equals recall.

It has also been argued that precision or recall are hard metrics that improperly measure the quality of results [30] [31] and that softer metrics such as the *distance ratio* metric proposed in [30] [32] would be more appropriate.

$$D = \frac{\sum_{\mathbf{x} \in \mathcal{A}(\mathbf{q})} d(\mathbf{x}, \mathbf{q})}{\sum_{\mathbf{x} \in \mathcal{G}(\mathbf{q})} d(\mathbf{x}, \mathbf{q})} \tag{7}$$

We compare against VA-LOW [29], a variant of the VA-Files that can return approximate NNs. In the VA-LOW, the search in the second phase is stopped once sufficient vectors have been visited to assure a certain precision. Figure 27 shows the performance of our clustering + reduced complexity hyperplane bound retrieval with precision as quality metric, while Figure 26 pertains to distance ratio.

We first note that even the very first cluster returns high-precision results. In order to reduce the number of sequential IOs, it is necessary to allow ≈300 clusters, which results in a few additional random disk IOs. On the other hand, in VA-LOW, several random and sequential disk access are necessary.

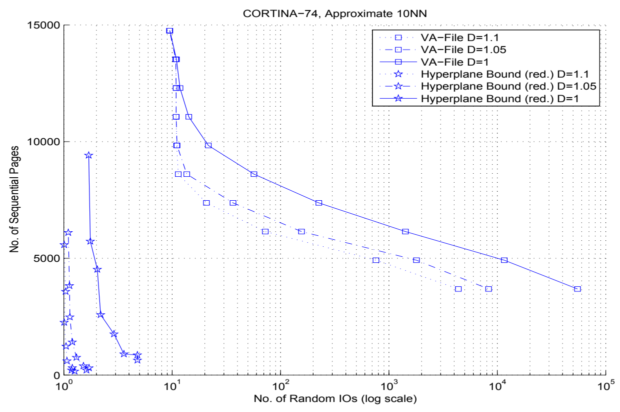

Fig. 26. Clustering vs. VA-LOW, Distance Ratio

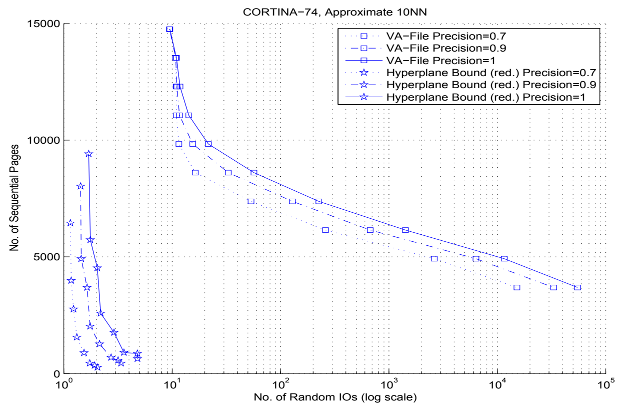

Fig. 27. Clustering vs. VA-LOW, Precision

Moreover, we observe that 100% precision results i.e. the retrieval of *exact* nearest neighbors, is possible with just the first few disk IOs. Of course, in this approach, there is no guarantee that the exact NNs have been found, even though experimentally we notice 100 % precision.

L. Adapting to Mahalanobis Distances

By proper user feedback, the Mahalanobis distance measure [17], [16], [5] can be updated and thus produce more relevant results. The database collects user feedback and periodically updates the distance measure and the index. The data-set is once again partitioned into Voronoi clusters $\{\mathcal{X}_m\}_1^K$ using the new distance measure (as explained in section VI). Since the boundaries between Voronoi clusters are still hyperplanes, we can still apply the hyperplane bounds to filter out irrelevant clusters. We note that to adapt the VA-File, iDistance and LDC to the new distance measure, we would definitely need to "whiten" or transform the data-set (to avoid any performance degradation).

We present the results from one such sample weight matrix. In our experiments, if d is dimensionality,

$$\Sigma = (\sum_{n=1}^{d} \lambda_n \mathbf{u}_n \mathbf{u}_n^T)^{-1} \tag{8}$$

where $\{\mathbf{u}_n\}$ was an orthonormal set of vectors (generated from a correlation matrix), with $1 \leq \lambda_n \leq 5, \forall n$ (following general guidelines established in [5]). For compactness, we present results only from the BIO-RETINA data-set (see Figure 28).

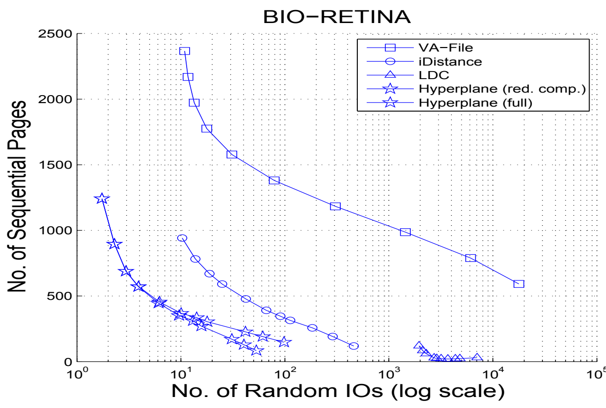

Fig. 28. IO Comparison - BIO-RETINA with Mahalanobis Distance

We note that the performance graph of the clustering plus hyperplane bounds is beneath the performance graphs of other indexes, with performance gains being largely retained. Thus, the hyperplane bound can be easily adapted to Mahalanobis distances.

IX. Conclusions

Real multidimensional data-sets exhibit significant correlations and non-uniform distributions. Hence, indexing with the VA-File, by performing uniform, scalar quantization, is suboptimal. We proposed an indexing method, based upon principles of vector quantization instead, where the data set is partitioned into Voronoi clusters and clusters are accessed in order of the query-cluster distances. We developed cluster-distance bounds based on separating hyperplane boundaries and our search index, complemented by these bounds, is applicable to Euclidean and Mahalanobis distance metrics. It obtained significant reductions in number of random IOs over several recently proposed indexes, when allowed (roughly) the same number of sequential pages, has a low computational cost and scales well with dimensions and size of the data-set.

We note that while the hyperplane bounds are better than MBR and MBS bounds, they are still loose compared with the true query-cluster distance (see Figures 6 and 7). Conceivably, the cluster-distance bounds can be further tightened, possibly by optimizing the clustering algorithm so as to optimize the cluster distance bounds. Future efforts would be directed toward this and other related problems.

References

[1] C. Faloutsos, *Searching in Multimedia Databases By Content*. Kluwer Academic Press, 1996.
[2] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is "nearest neighbor" meaningful?" in *ICDT*, 1999, pp. 217–235.
[3] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional spaces," in *ICDT*, 2001, pp. 420–434.
[4] B. U. Pagel, F. Korn, and C. Faloutsos, "Deflating the dimensionality curse using multiple fractal dimensions," in *ICDE*, 2000, pp. 589–598.
[5] T. Huang and X. S. Zhou, "Image retrieval with relevance feedback: From heuristic weight adjustment to optimal learning methods," in *ICIP*, vol. 3, 2001, pp. 2–5.
[6] J. Davis, B. Kulis, P. Jain, S. Sra, and I. Dhillon, "Information-theoretic metric learning," in *ICML*, 2007, pp. 209–216.
[7] R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces." in *VLDB*, August 1998, pp. 194–205.
[8] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992.
[9] T. Zhang, R. Ramakrishnan, and M. Livny, "BIRCH: An efficient data clustering method for very large databases." in *SIGMOD*, 1996, pp. 103–114.
[10] N. Katayama and S. Satoh, "The SR-tree: An index structure for high-dimensional nearest neighbor queries." in *SIGMOD*, May 1997, pp. 369–380.
[11] D. A. White and R. Jain, "Similarity indexing with the SS-tree," in *ICDE*, 1996, pp. 516–523.
[12] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: an efficient and robust access method for points and rectangles," in *SIGMOD*, 1990, pp. 322–331.
[13] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-tree: An index structure for high-dimensional spaces using relative approximation," in *VLDB*, September 2000, pp. 516–526.
[14] S. Berchtold, C. Bohm, H. V. Jagadish, H. P. Kriegel, and J. Sander, "Independent Quantization: An index compression technique for high-dimensional data spaces," in *ICDE*, 2000, pp. 577–588.
[15] C. Yu, B. C. Ooi, K. L. Tan, and H. V. Jagadish, "Indexing the distance: An efficient method to knn processing." in *VLDB*, September 2001, pp. 421–430.
[16] Y. Ishikawa, R. Subramanya, and C. Faloutsos, "Mindreader: Querying databases through multiple examples," in *VLDB*, August 1998, pp. 218–227.
[17] Y. Rui and T. Huang, "Optimizing learning in image retrieval," *CVPR*, vol. 1, pp. 1236–1243, 2000.
[18] S. Ramaswamy and K. Rose, "Adaptive cluster-distance bounding for similarity search in image databases," in *ICIP*, vol. 6, 2007, pp. 381–384.
[19] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung, "LDC: Enabling search by partial distance in a hyper-dimensional space." in *ICDE*, 2004, pp. 6–17.
[20] A. Guttman, "R-trees: A dynamic index structure for spatial searching." in *SIGMOD*, 1984, pp. 47–57.
[21] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces." in *VLDB*, 1997, pp. 426–435.
[22] R. Bellman, *Adaptive Control Processes: A Guided Tour*. Princeton, NJ: Princeton University Press, 1961.
[23] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi, "Vector approximation based indexing for non-uniform high dimensional data sets," in *CIKM*, 2000, pp. 202–209.
[24] K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: A new approach to indexing high dimensional spaces." in *VLDB*, September 2000, pp. 89–100.
[25] K. Vu, K. Hua, H. Cheng, and S. Lang, "A non-linear dimensionality-reduction technique for fast similarity search in large databases," in *SIGMOD*, 2006, pp. 527–538.
[26] S. Berchtold, C. Bohm, and H. Kriegel, "The Pyramid-technique: Towards breaking the curse of dimensionality," in *SIGMOD*, 1998, pp. 142–153.
[27] H. Jin, B. C. Ooi, H. T. Shen, C. Yu, and A. Zhou, "An adaptive and efficient dimensionality reduction algorithm for high-dimensional indexing." in *ICDE*, March 2003, pp. 87–98.
[28] P. Ciaccia and M. Patella, "PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces," in *ICDE*, 2000, pp. 244–255.
[29] R. Weber and K. Böhm, "Trading quality for time with nearest neighbor search." in *EDBT*, 2000, pp. 21–35.
[30] E. Tuncel, H. Ferhatosmanoglu, and K. Rose, "VQ-Index: An index structure for similarity searching in multimedia databases." in *ACM Multimedia*, 2002, pp. 543–552.
[31] E. Tuncel and K. Rose, "Towards optimal clustering for approximate similarity searching." in *ICME*, vol. 2, August 2002, pp. 497–500.
[32] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi, "Approximate nearest neighbor searching in multimedia databases." in *ICDE*, April 2001, pp. 503–511.
[33] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing." in *VLDB*, September 1999, pp. 518–529.
[34] P. Ciaccia and M. Patella, "Approximate similarity queries: A survey." *CSITE-08-01 Technical Report*, May 2001.

[35] E. Tuncel, P. Koulgi, and K. Rose, "Rate-distortion approach to databases: Storage and content-based retrieval," *IEEE Trans. on Information Theory*, vol. 50, no. 6, pp. 953–967, 2004.
[36] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: John Wiley & Sons, 1996.

ACKNOWLEDGMENT

The authors would like to thank Dr. Mark Verardo, Prof. Steven Fisher and Dr. Geoffrey Lewis, from Neuroscience Research Institute and Center for Bio-Image Informatics, UCSB for providing the image data, Dr. Jiyun Biyun and Rajesh Kumar, of the Center for Bio-Image Informatics, UCSB, who extracted the features for the BIO-RETINA data-set and JieJun Xu, of the Vision Research Lab, UCSB, who extracted the features for the CORTINA data-set.

Sharadh Ramaswamy Sharadh Ramaswamy received the B.Tech and M.Tech degrees in 2003 from the Indian Institute of Technology, Madras and the Ph.D degree in 2009 from the University of California, Santa Barbara. He then joined MayaChitra Inc. as a member of Research Staff. His research interests lie in compression, sensor networks, search and retrieval in databases, video processing and machine learning.

Kenneth Rose Kenneth Rose (S'85-M'91-SM'01-F'03) received the Ph.D. degree in 1991 from the California Institute of Technology. He then joined the Department of Electrical and Computer Engineering, University of California at Santa Barbara, where he is currently a Professor. His main research activities are in the areas of information theory and signal processing, and include rate-distortion theory, source and source-channel coding, audio and video coding and networking, pattern recognition, search and retrieval in databases, and nonconvex optimization. He is interested in the relations between information theory, estimation theory, and statistical physics, and their potential impact on fundamental and practical problems in diverse disciplines. He was co-recipient of the 1990 William R. Bennett Prize Paper Award of the IEEE Communications Society, as well as the 2004 and 2007 IEEE Signal Processing Society Best Paper Awards.