Vector Approximation based Indexing for Non-uniform High Dimensional Data Sets

Hakan Ferhatosmanoglu
Department of Computer Science

Ertem Tuncel
Department of Electrical and Computer Engineering

Divyakant Agrawal
Department of Computer Science

Amr El Abbadi
Department of Computer Science

University of California
Santa Barbara, CA 93106
{hakan,agrawal,amr}@cs.ucsb.edu, ertem@laurel.ece.ucsb.edu

Abstract

With the proliferation of multimedia data, there is increasing need to support the indexing and searching of high dimensional data. Recently, a vector approximation based technique called VA-file has been proposed for indexing high dimensional data. It has been shown that the VA-file is an effective technique compared to the current approaches based on space and data partitioning. The VA-file gives good performance especially when the data set is uniformly distributed. Real data sets are not uniformly distributed, are often clustered, and the dimensions of the feature vectors in real data sets are usually correlated. More careful analysis for non-uniform or correlated data is needed for effectively indexing high dimensional data. We propose a solution to these problems and propose a new technique for indexing high dimensional data sets based on vector approximations. We conclude with an evaluation of nearest neighbor queries and show that the proposed technique results significant improvements over the current VA-file approach for several real data sets.

1 Introduction

As modern databases increasingly integrate various types of information, such as multimedia data, it becomes necessary to support efficient retrieval in such systems. Example of such applications include Multimedia Information Systems [11], CAD/CAM [6], Geographical Information Systems (GIS) [9], time-series databases [12], medical imaging [10]. The data is usually represented by a *feature vector* which summarizes the original data with some number of dimensions. The similarity between two objects is defined with a distance function, e.g., Euclidean distance, between the corresponding feature vectors. A well-known type of query is the similarity query. For example, in image databases, the user may pose a query asking for the most similar images to a given image. Similarity query with multi-dimensional data is usually implemented by finding the k closest feature vectors to the feature vector of the query data. Since the amount of data that needs to be processed is increasing rapidly, it becomes necessary to support techniques for efficient similarity searching in modern databases. Commercial data warehouses are doubling their size every 9-12 months and satellite data repositories will soon add one to two terabytes of data in a day [1]. If the current trends continue, large organizations will have petabytes of data that need to be retrieved and processed for various purposes such as data mining and decision support [7].

With the proliferation of multimedia data, several applications need to be supported for indexing and searching of high dimensional data. In some applications, such as GIS, the feature vectors usually have small number of dimensions, typically 2 dimensions. Numerous index structures exist that facilitate search and retrieval of two dimensional data or spatial databases [30, 13, 28, 29, 16, 2]. The general approach for high dimensional indexing was to extend the spatial index structures and to propose new ones to deal with the high dimensional nature of information [20, 5, 32, 19, 24, 17, 8]. In fact, Berchtold et al. [3, 4] have argued that using these multi-dimensional index structures for searching beyond a certain dimension becomes worse than a sequential scan.

Weber et al. [31] have developed a quantitative analysis and performance study of similarity search techniques for high dimensional data sets. They formally show that for data sets with *uniform distribution*, the indexing techniques based on partitioning and clustering are outperformed on average by a simple sequential scan if the number of dimensions exceeds around 10. Under the assumption of uniformity and independence, they formally establish that there is no organization of high dimensional vector space based on partitioning or clustering which does not degenerate to a sequential scan if dimensionality exceeds a certain threshold. Instead of the data space or data partitioning, or clustering based approaches which suffer from the dimensionality curse, a new approach based on *vector approximations* is proposed which overcomes these problems and scales well even if dimensionality increases. The technique proposed in [31] is called VA-File, i.e., Vector Approximation File, and the technique has been shown to be very effective compared to current indexing techniques. The biggest advantage of the technique is that it does not suffer from the dimensionality problem.

Although it provides significant improvements compared to current techniques, the VA-file itself suffers from various problems. In this paper, we discuss these problems and propose a solution that renders the vector approximation idea more effectively. The major problems in the current VA-file that need to be overcome can be summarized as the assumption of independent or uncorrelated dimensions, uniform bit allocation, and the simple partitioning technique. The dimensions of the feature vectors in a real data set usually are dependent or correlated. Moreover, real data sets are not uniformly distributed. More careful analysis for non-uniform or correlated data is needed for effectively indexing high dimensional data. We discuss the problems that the VA-file suffers from if the data set is not uniformly distributed, especially if it is highly correlated or clustered. We propose a new technique which is much more suitable for non-uniform data sets.

Section 2 discusses the indexing technique based on vector approximations and discusses the importance of approximation quality in such a technique. In Section 3, we motivate the need for a technique for non-uniform data sets and propose a new technique that involves a careful analysis for non-uniform or correlated data. In Section 4, we discuss the impact of better approximations to the efficiency using several illustrative examples. In Section 5, we perform an evaluation of nearest neighbor queries and show that the proposed technique results in significant improvements over the current VA-file approach for several real data sets. Conclusions appear in Section 6.

2 Indexing Based on Vector Approximation

The VA-file approach divides the data space into 2^b rectangular cells where b is the total number of bits specified by the user [31]. Each dimension is allocated a number of bits, which are used to divide it into equal populated cells on that dimension. Each cell has a bit representation of length b which approximates the data points that fall into a cell by the corresponding bit representation of the cell. The VA-file itself is simply an array of these bit vector approximations based on the quantization of the original feature vectors. An example of a regular VA-file partition of the data space is given in Figure 1. The two dimensional data set shown is actually created by taking two dimensions of feature vectors of real time-series data which contains stock price movements of 2000 companies. For simplicity, each dimension is assigned 2 bits, and hence each dimension is divided into 4 intervals of equal population.

Nearest neighbor searching in a VA-file has two major phases [31]. In the first phase, the set of all vector approximations is scanned sequentially and lower and upper bounds on the distance of each vector to the query vector are computed. The real distance can not be smaller than the distance between the query vector and the closest point on the corresponding cell. Therefore, the real distance can be lower-bounded by this smallest possible distance. Similarly, the distance of the furthest point in the cell with the query point determines the upper bound of the real distance. In this phase, if an approximation is encountered such that its lower bound exceeds the smallest upper bound found so far, the corresponding objects can be eliminated since at least one better candidate exists. At the end of the first phase, the vectors with the smallest bounds are found to be the *candidates* for the nearest neighbor of the query. In the second phase, the algorithm traverses the real feature vectors that correspond to the candidate set remaining after filtering. The real feature vectors of the candidates are visited until it is guaranteed that the actual nearest neighbor is found. The feature vectors are visited in the order of their lower bounds and then exact distances to the query point are computed. One of the advantages of traversing the feature vectors in the order of their lower bounds to the query is that the algorithm can stop at a certain point and does not check the rest of the candidates. If a lower bound is reached that is greater than the k-th actual nearest neighbor distance seen so far, then the algorithm stops retrieving the rest of the candidates. It is important

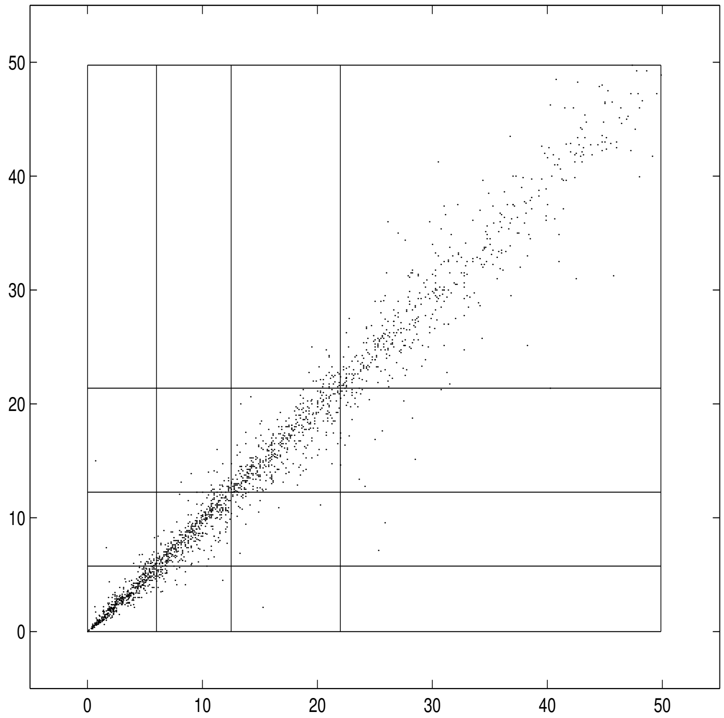


Figure 1. Regular VA-file partition of the 2 dimensional data set. Each dimension is divided into 4 equally populated intervals.

to note that the accesses in the second phase are secondary storage accesses. It is crucial to decrease the number of vectors visited in the second phase to reduce the number of random I/O that must be incurred during the nearest neighbor search.

In the first phase of the algorithm, the filtering is performed based on the lower and the upper bounds of the distance of the data point from the query point. The performance of the VA-File approach heavily depends on the quality of these bounds that are computed in the first phase. The lower and upper bounds are important for the quality of the approximations of the data points. Better approximations lead to more filtering and therefore faster queries.

In particular, the quality of the lower bound directly effects the number of vectors visited in the second phase where the feature vectors are traversed in the order of their lower bounds to the query point. The nearest neighbor algorithm stops at a point where a lower bound is reached that is greater than the k-th actual nearest neighbor distance seen so far. Tighter lower bounds cause the algorithm to stop earlier and visit less number of candidates, and therefore less number of page accesses from the secondary storage are occurred. The majority of these page accesses perform random seeks. Therefore, better lower bounds are needed to reduce the random I/O which dominates the query time. Together with tight upper bounds, the better lower bound also means less number of candidates remaining for the second phase. In the first phase, the pruning is based on the lower and upper bounds on each distance. An object is eliminated if its approximation exceeds the smallest upper bound found so far. If the lower and upper bounds on the distances are tight and therefore close to each other, more elimination is performed in the first phase of the algorithm. In other words, tighter lower and upper bounds also mean less number of candidates remaining for the second phase of the VA-file algorithm.

3 Vector Approximation Based Indexing for Non-Uniform Data Sets

In the regular VA-file approach, there is an implicit assumption that the dimensions are independent, or at least uncorrelated, because each dimension is divided into cells independently. Also, although there is always an option of non-uniform bit allocation among dimensions, no specific algorithm for that option was proposed. Finally, each dimension i is divided into 2^{b_i} cells of either equal size, or equal population, which are the two simplest partitionings that coincide for a uniformly distributed data. In this section, we mention the major problems that need to be overcome and propose approaches that lead to more efficient searching using vector approximation files. In particular, we highlight some of the problems that VA-files suffer if the data set is not uniformly distributed, especially when it is highly correlated or clustered. We propose a new technique which is much more suitable for such data sets.

3.1 Motivation

A scalar quantizer [14] is a VA-file together with representative values assigned to each cell. The target is to achieve the least reproduction error, i.e., the least average Euclidean distance between data points and their representatives. The partitioning performed by the VA-file approach can be viewed as a scalar quantization, except that the approach does not care about the representative values for the cells. We claim that a scalar quantization designed by directly aiming for the least possible reproduction error that would result in much tighter lower and upper bounds for the distances between the query point and the data points. As discussed in the previous section, tighter lower bounds mean less number of vectors visited in the second phase of the VA-file algorithm, and tighter upper bounds mean better filtering of data in the first phase.

A better approach for designing a scalar quantizer should follow these steps [14]:

1. The data should be transformed into a more suitable domain,
2. In general, the total bits in the quota should be allocated among the transformed dimensions non-uniformly,
3. An optimal scalar quantizer should be designed for each transformed dimension independently with the allocated number of bits. In general, the quantizers should not assume uniform data and should make use of the data statistics.

3.2 VA-file in KLT Domain

It is a well known fact that better scalar quantization performance is achieved when the dimensions of the data are uncorrelated. So, if the dimensions are highly correlated as in the example shown in Figure 1, or even slightly correlated, in general it is a good idea to decorrelate them by applying a unitary transform to the data set. A transform is called unitary if it preserves all the "angles and lengths" in the data space.

If the process generating the data set is known, a fixed transform might be suitable. For example, if we have a time series data, the Discrete Fourier Transform (DFT), the Discrete Cosine Transform (DCT), or the Discrete Wavelet Transform (DWT) will be very good choices, because they will be successful in decorrelating the data. However, in general, one has to estimate the correlation matrix for the given data set, and derive the unique decorrelating transform for it. The resultant transform is known as the Karhunen Loeve Transform (KLT) [18, 23].

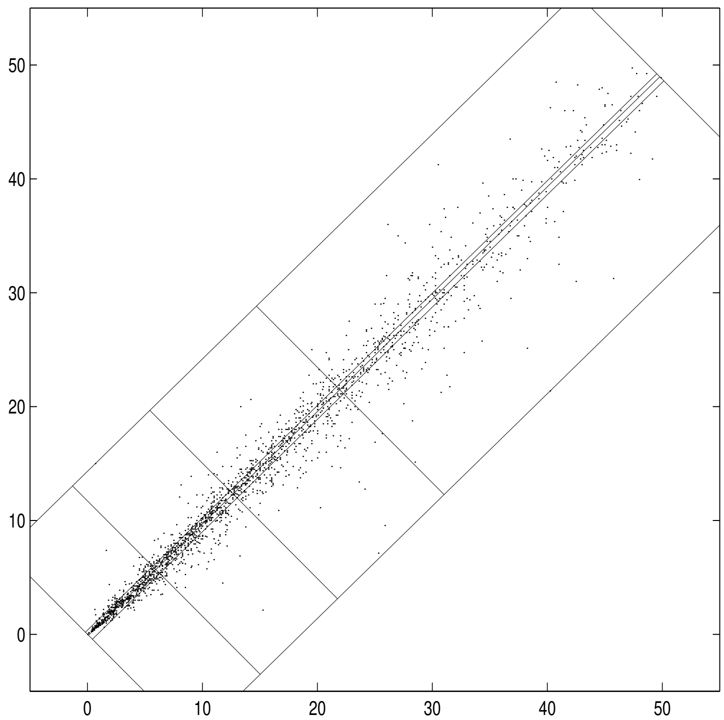


Figure 2. VA-file partition of the same data set after applying KLT. Each KLT-dimension is divided into 4 equally populated intervals.

For the example given in Figure 1, KLT will basically result in a 45-degree rotation, because the dimensions are almost maximally correlated, i.e., the data is almost spread through a straight line with slope 1. In Figure 2, we present the idea of creating the VA-file after transforming the data set using KLT. For the sake of being able to compare the result with Figure 1, the data set and also the

VA-file are both rotated back using the inverse KLT.

The VA-file approach depends on the bit allocation that is performed to each dimension independently. When some of the dimensions are correlated an independent bit allocation on correlated dimensions does not capture some important information about the data point distribution. However, an independent bit allocation should be based on independent dimensions. KLT attempts to transform the data set to a new set of dimensions that reduce the correlation as much as possible. This problem can be seen by comparing Figures 1 and 2. The VA-file creates cells in non-transformed domain based on having equal population on the x and y dimensions (Figure 1). This direct application of the VA-file does not capture the following important property of the data set: the data set is correlated and spread through an axis that is in some way a combination of x and y. In this example, it handles the data set as if it is uniformly distributed. Several boxes in Figure 1 have similar square-like area and the cells around the diagonal of the data space overflow, while in Figure 2 the data objects are more equally distributed among the cells.

Since the efficiency of the vector approximation based indexing technique depends on the approximation quality of the cells, it is important to improve this by more careful analysis on the dimensions. Although decorrelating the dimensions helps this problem, the advantage of KLT becomes much more apparent when it is used as a preprocessing step for a more accurate analysis of the data set. The KLT transforms the data set into a more suitable domain where the following steps of our algorithm, i.e., non-uniform bit allocation and the scalar quantization, are able to obtain even more advantages from the statistical properties of the data set.

3.3 Non-uniform Bit Allocation Algorithm

It is obvious from Figure 2 that allocating the bits in our quota, i.e., the total number of bits that is allowed for approximating the data vectors, non-uniformly among the KLT-dimensions would result in better quantization performance. For example, if we allocate 4 bits to the major axis of the data, and no bits to the minor axis, and thereby keeping the total as 4 bits, we obtain the VA-file shown in Figure 3. The information that is stored in each dimension of the feature vector varies depending on the feature vectors. The goal is to approximate this information with a minimum number of bits with maximum accuracy. Therefore, for several cases it is crucial to analyze the dimensions and allocate the bits to dimensions, rather than the simplistic uniform bit allocation, so that the resulting accuracy obtained from the approximation is maximized.

Non-uniform bit allocation is an effective way to increase the accuracy of the approximations for any data set. However, it becomes more advantageous and more feasible in the KLT domain. The difference between the amount of energy stored in each dimension becomes much more when dimensions are transformed using KLT. KLT is known to have maximum energy compaction property for any given data set. That means KLT successfully accumulates the data energy more into the first dimensions.

For the example data set (Figures 2 and 3), it can be seen that in the KLT domain the y dimensions

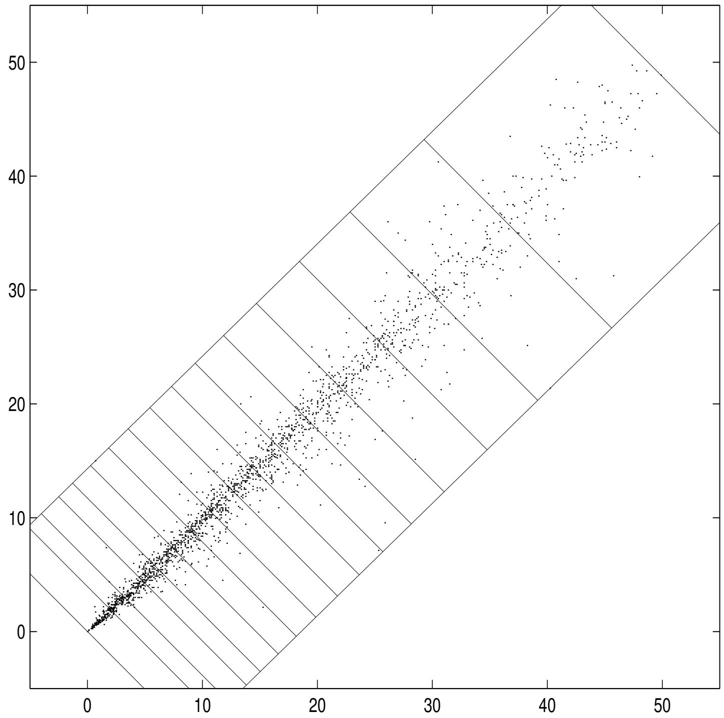


Figure 3. VA-file partition of the same data set after changing the bit allocation. Now, the major axis has 16 intervals, and the minor axis has only 1 interval.

of the points are already populated in the middle and close to each other, while the x dimensions of the data points are spread throughout the axis. To differentiate and therefore better approximate the x dimension of the points, non-uniform bit allocation allocates more bits to this axis while the total bit quota is kept same as before. With uniform bit allocation, the approximation quality of the cells would decrease since the points that are far away are allocated to the same cell.

Let the variance of dimension i be σ_i^2, and the number of bits allocated to dimension i be b_i. We have a quota of b bits, i.e, it must be true that

$$b = \sum_i b_i \ .$$

In quantization theory, it is a rule of thumb that, if $\sigma_i^2 \geq 4^k \sigma_j^2$, then $b_i \geq b_j + k$ [14]. In fact, this rule is based on the assumption of high resolution uniform quantization, i.e., equal-sized intervals and $b_i, b_j \gg 1$. Nevertheless, even when we do not have so many bits, and we do not perform uniform quantization, this rule still proves to be a good heuristic. Hence, the resultant bit allocation algorithm for VA-file is derived based on the rule of thumb as follows:

1. Begin with $d_i = \sigma_i^2$, $b_i = 0$ for all i, and $k = 0$.
2. Let $j = \max_i^{-1} d_i$. Then increment b_j and $d_j \leftarrow d_j/4$.

3. Increment k. If $k < b$, go to 2, else stop.

Figure 3 is the result of applying the above bit allocation algorithm distributing the 4 bits among 2 dimensions.

3.4 Optimal Quantization

Once we know how many bits are to be allocated to a given KLT-dimension, we can design an actual scalar quantizer for it, rather than dividing the dimension into equally populated or equal-sized intervals. A quantizer is characterized by the decision intervals, and the corresponding representative values for each interval. Even though we are not going to make use of the representative values directly, they also have to be designed together with the decision intervals. A very popular algorithm for designing a quantizer with K intervals is Lloyd's algorithm [22], which is nothing but the K-means algorithm for clustering [25, 21], specialized to one dimension:

> Denote by t_n the value of the nth data point along the objective KLT-dimension. Start with a given set of intervals $[c_i, c_{i+1})$ for $i = 1, \ldots, K$, where $c_1 = \min_n t_n$ and $c_{K+1} = \epsilon + \max_n t_n$. Set $\Delta = \infty$, and fix $\gamma > 0$. Denote by r_i the representative value for the interval i.
>
> 1. For $i = 1, \ldots, K$, compute the new representative value r_i as the center of mass of all data points in the interval $[c_i, c_{i+1})$, i.e.,
>
> $$r_i = \frac{1}{N_i} \sum_{t_n \in [c_i, c_{i+1})} t_n \ ,$$
>
> where N_i is the total number of data points in the interval $[c_i, c_{i+1})$.
>
> 2. Compute the new intervals by
>
> $$c_i = \frac{r_{i-1} + r_i}{2}$$
>
> for $i = 2, \ldots, K$.
>
> 3. Compute the total representation error as
>
> $$\Delta' = \sum_{i=1}^{K} \sum_{t_n \in [c_i, c_{i+1})} (t_n - r_i)^2 \ .$$
>
> If $\frac{\Delta - \Delta'}{\Delta} < \gamma$, then STOP. Otherwise set $\Delta = \Delta'$, and go to step 1.

The above algorithm is guaranteed to converge. However, it may get trapped into poor local minima, and hence needs clever initialization. For example, to initialize the algorithm with the equally-populated cells, i.e., the current VA-file approach, would be a good idea. The approximation quality of the cells is improved in each step of the algorithm. By doing so, we obtained a resultant VA-file as shown in Figure 4. After applying this third step, the error introduced by approximating the data points by the corresponding cells is further reduced. Since the representation quality of the cells directly effects the quality of the VA-file, this leads to better pruning both in the first and the second phase of searching. The improvements by the proposed technique is much more apparent for higher dimensional data sets than the two-dimensional example illustrated in this section. These results are also supported later in the paper by the analysis of the performance results of each technique.

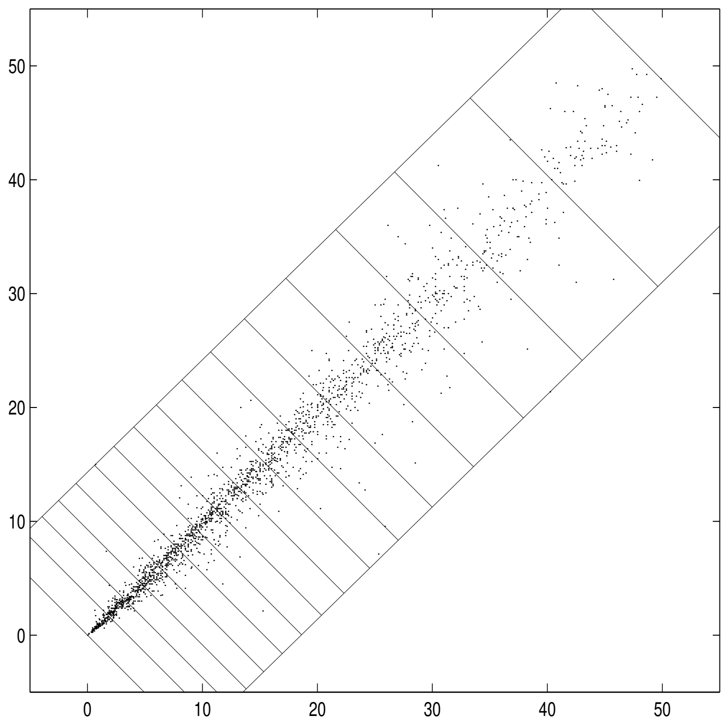
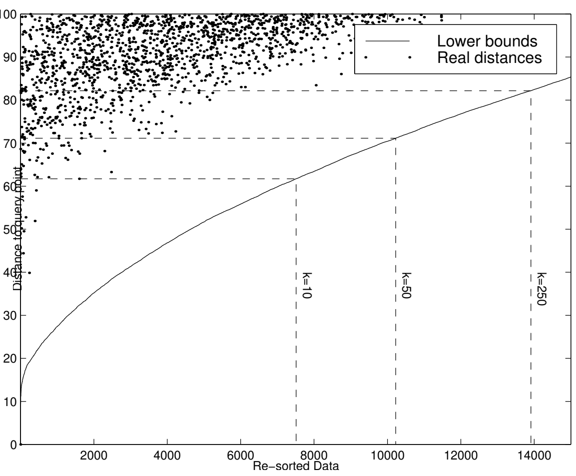


Figure 4. VA-file partition of the same data set after applying Lloyd's algorithm.

4 Illustrative Examples on the Importance of Approximation Quality Improvements

In this section, we illustrate the impact of tighter bounds and of the approximation quality improvements over the regular VA-file algorithm. To clarify the importance of the bounds, the discussions are done for an illustrative query. However, as will be seen in the performance evaluation section, the improved VA-file keeps its superior performance on the average over the regular approach. For this section, we have used a 100,000 60-dimensional vectors representing texture feature vectors of Landsat images [27]. Texture information of blocks of large aerial photographs are computed using Gabor filtering. The Gabor filter bank consists of 5 scales and 6 orientations of filters, therefore the total number of filters is $5 \times 6 = 30$. The mean and standard deviation of each filtered output are used to create the feature vector. Therefore the dimensionality of each vector becomes $30 \times 2 = 60$.

This data set provides challenging problems in high dimensional indexing and is widely used in high dimensional indexing and similarity searching research [26, 15]. For this section, we assumed a total bit quota of 180 bits, i.e., 3 bits per dimension.



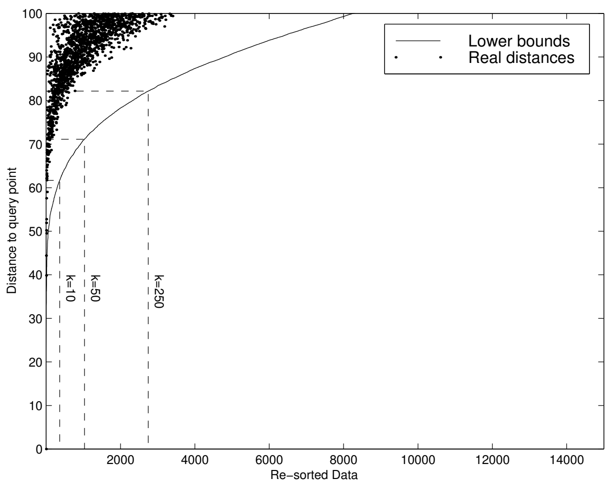


Figure 5. Comparison between the regular (left) and the improved (right) VA-file methods in terms of tightness of lower bounds and number of vectors visited.

In Figure 5, the tightness of the lower bounds achieved by the regular and the improved VA-file algorithms are compared using an example. In both cases, lower bounds are calculated for the same typical query point, and the data is re-sorted in ascending order of lower bound, because this is exactly order the vectors are visited in the second phase. The x-axis represents the data objects in resorted order and the y-axis represents the distances to the given query point. The dots in the figure therefore correspond to the *actual* distances between the re-sorted data points and the query point. Since the sorting is done according to the lower bounds, the plot for the corresponding lower bounds for each data point form a nice curve, instead of scattered dots, below the actual distances. We compare the number of vectors visited for k-NN queries, where $k = 10$, $k = 50$, and $k = 250$. The algorithm has to visit all vectors with lower bounds less than the distance of the kth nearest data. The intercepts of dashed lines on the Y and X axes respectively correspond to the distance of the kth nearest data, and the number of vectors visited. As shown in the figure, as the lower bounds get tighter, the data becomes more suitably sorted, i.e., more or less in an ascending order of real distances to the query point. Hence, the actual k neighbors are found by visiting less number of vectors (=accessing less number of pages). For example, for this query point with $k = 10$, the kth nearest neighbor has a distance around 62, and the regular VA-file method has to visit 15 times as many vectors as the improved method does.

Figure 6 shows that tighter lower bounds together with tighter upper bounds imply better filtering

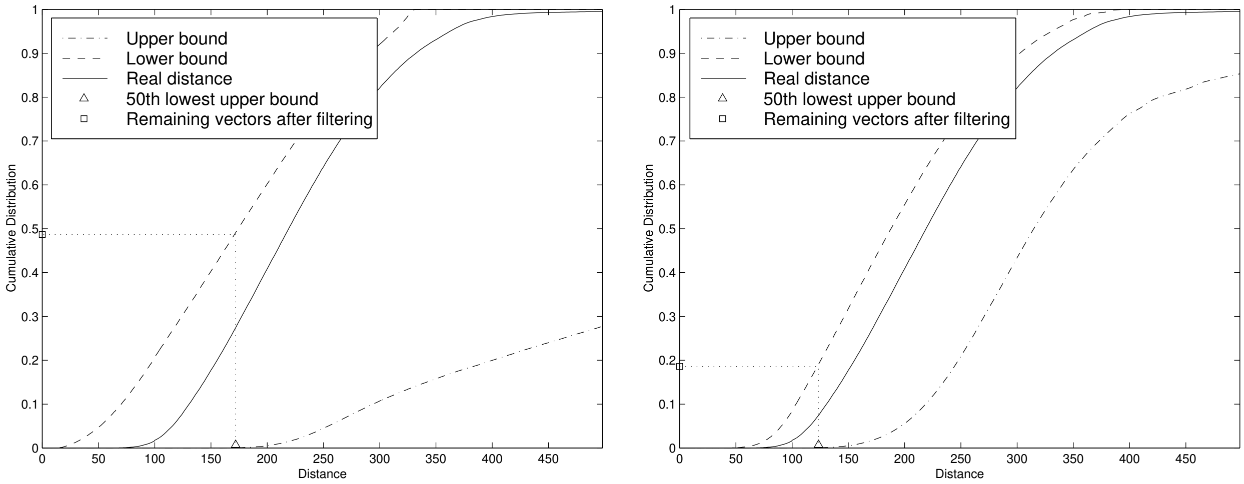


Figure 6. Comparison between the regular (left) and the improved (right) VA-file methods, in terms of the cumulative distributions of lower and upper bounds, and number of vectors eliminated in the first phase.

in the first phase of the k-NN algorithm. To illustrate this important property, the same query point as before is chosen. Cumulative distributions of real distances, lower bounds, and upper bounds are shown together for both the regular and the improved VA-file methods. A cumulative distribution is defined as the integral (or cumulative sum) of a regular probability density function, and is more appropriate to consider for our purposes here. For example, if the lower bound cumulative distribution takes the value 0.2 for distance 25, it means that one fifth of the data has lower bounds less than or equal to 25.

For a k-NN query, all vectors having lower bounds less than the kth lowest upper bound remain unfiltered in the first phase. In fact, if the data were sorted in the best possible order, only those vectors would have remained after the filtering. In the figures, k is set to 50, and the triangle on the X-axis shows the value of the 50th lowest upper bound, and the square on the Y-axis shows the fraction of vectors that remained unfiltered for this best case scenario. Our experiments show that the actual fraction of unfiltered vectors are very close to the best-case value and not plotted here for the sake of clarity. Tighter upper bounds reduce the value of the 50th lowest upper bound, and hence reduce the fraction of the vectors that remain unfiltered after the first phase. On the other hand, tighter lower bounds also reduce the same fraction, because less vectors would have lower bounds less than the 50th lowest upper bound. This phenomenon is clearly observed from Figure 6, where the improved VA-file method displayed on the right has much tighter lower and upper bounds for the fixed query point than the regular VA-file method displayed on the left.

In Figure 7, the cumulative distributions of real distances, lower bounds, and upper bounds of both methods are shown on the same plot. Also shown by circles are the fraction of vectors visited

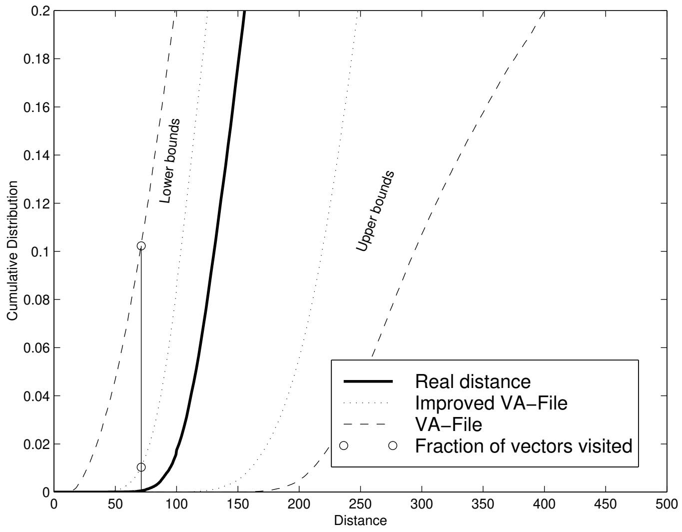


Figure 7. Comparison between the regular and the improved VA-file methods, in terms of the cumulative distributions of lower and upper bounds, and number of vectors visited in the second phase.

in the second phase, again for both algorithms. The fraction of vectors visited corresponds to the value the lower bound cumulative distribution attains for the 50th lowest distance. As seen from the figure, the regular VA-file method visits around $100,000 \times 0.09 = 9000$ vectors, whereas the improved method visits only $100,000 \times 0.015 = 1500$ vectors.

5 Performance Evaluation

We used real-life data sets for the experimental evaluation. The first one is the same 60-dimensional satellite image feature data set of size 100,000. The detailed description of this data set was given in the previous section. The second one is a 64-dimensional color image histogram data set of size 12,000. The vectors represent color histograms computed from a commercial CD-ROM. We have designed regular and improved VA-files for 3, 4, 5, and 6 bits/dimension. In other words, the total number of bits allocated to the feature vectors are kept same for each technique. We performed queries asking 10-NN of the data points in the data set. The same NN algorithm that was discussed before was applied to both techniques. The performance of the methods will be evaluated in terms of their first phase elimination power, and the number of vectors they have to visit in the second phase. For all cases the stated results are mean values. Better elimination power in the first phase leads to less number of candidates remaining for the second phase of the algorithm and less number of vectors visited in the second phase means less number of I/O operations needed.

Figure 8 compares the first and second phase performances of the methods, for the satellite image data. Note that the figures are plotted on a logarithmic Y-axis. In the first phase, for the regular

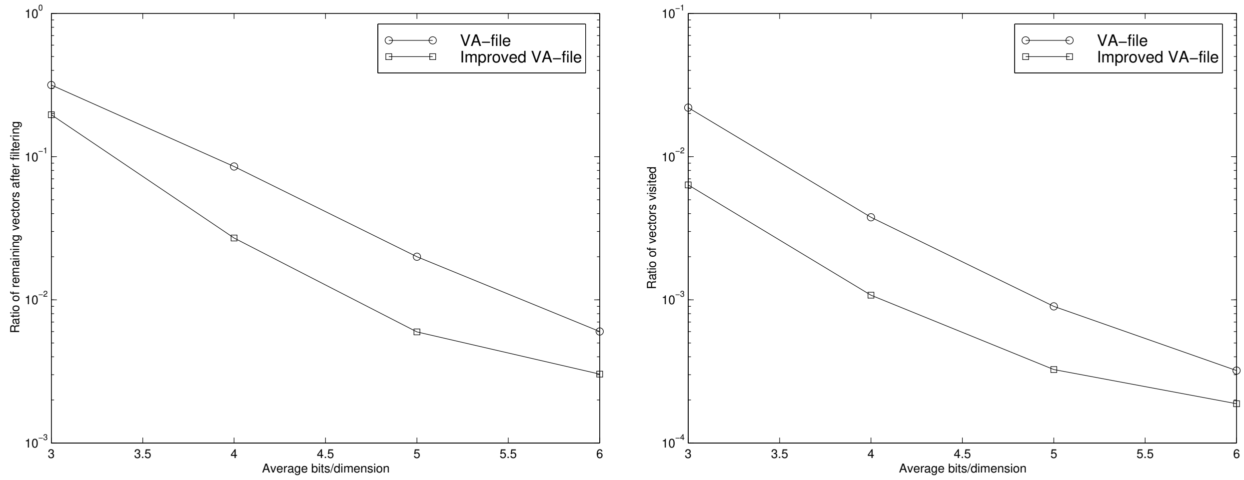


Figure 8. Comparison between the regular and the improved VA-file methods, in terms of the first (left) and second (right) phase performances, for the satellite image data.

VA-file method, there are 1.5 to 3 times as many remaining vectors after filtering as there are for the improved VA-file method. In the second phase, the regular VA-file method visits 1.7 to 3.5 times as many vectors as the improved VA-file method does. The results in the second phase can be seen as directly proportional to the number of page accesses and I/O cost during the search. Note that any VA-file method has to visit at least 10 vectors, i.e., 10^{-4} of all the data. For example, by just reading 9 more vectors for the 6-bit case, the improved VA-file achieves the task, whereas the regular VA-file needed to visit 22 more.

Figure 9 shows the comparison of the two methods for the color image histogram data. In the first phase, for the regular VA-file method, there are 1.5 to 5 times as many remaining vectors after filtering as there are for the improved VA-file method. In the second phase, the regular VA-file method visits 2.2 to 8.2 times as many vectors as the improved VA-file method does. This means that the I/O cost occurred during the k-nearest neighbor searching in improved VA-file is much less than the regular approach. In this case, any VA-file method has to visit at least 10 vectors, i.e., 8.3×10^{-4} of all the data. So by just reading 4 more vectors for the 6-bit case, the improved VA-file achieves the task, whereas the regular VA-file needed to visit 20 more.

6 Conclusion

In this paper, we proposed a new vector approximation based indexing technique for non-uniform high-dimensional data sets. We stated and discussed the problems associated with the VA-file which is an effective technique for indexing high dimensional data. The current VA-file approach assumes independent or uncorrelated dimensions, applies uniform bit allocation, and is based on a simple partitioning technique. More careful analysis for non-uniform or correlated data is needed for ef-

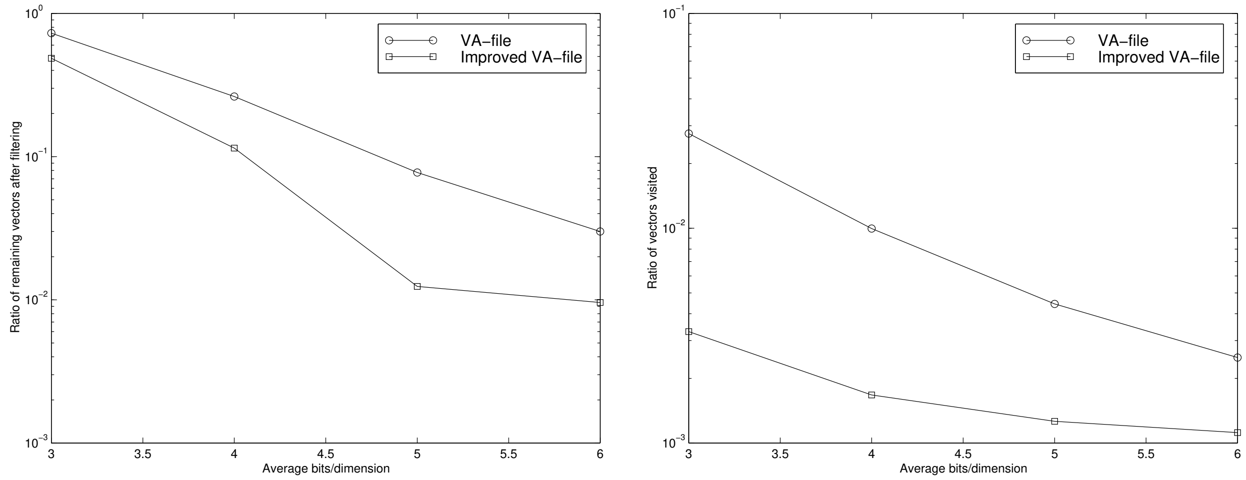


Figure 9. Comparison between the regular and the improved VA-file methods, in terms of the first (left) and second (right) phase performances, for the color image histogram data.

fectively indexing real data sets. Real data sets are not uniformly distributed, are often clustered, and the dimensions of the feature vectors in real data sets are usually correlated. We proposed a solution to these problems and proposed a new technique for indexing of non-uniform high dimensional data sets based on vector approximations. In particular, our technique consists of three major steps, (i) KLT to transform the data into a more suitable domain, (ii) an algorithm for non-uniform bit allocation, and (iii) an optimal quantization that makes use of the data statistics. With illustrative examples, we demonstrate the impact of the lower and upper bounds on the efficiency of nearest neighbor searching. The proposed technique achieves significantly better approximations of the distances of the points to the queries, i.e. better lower and upper bounds. We conclude with a performance evaluation of nearest neighbor queries and show that the proposed technique results significant improvements on the I/O cost of queries over the current VA-file approach for several real data sets.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active disks: Programming model, algorithms and evaluation. In *ASPLOS-VIII*, pages 81–91, September 1998.

[2] 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.

[3] 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*, Tuscon, Arizona, 1997.

[4] 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.

[5] S. Berchtold, D. A. Keim, and H. P. Kreigel. The X-tree: An index structure for high-dimensional data. In *22nd. Conference on Very Large Databases*, pages 28–39, Bombay, India, 1996.

[6] Kriegel H.-P. Berchtold S. S3: Similarity search in cad database systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 564–567, Tuscon, Arizona, 1997.

[7] P. Bernstein, M. Brodie, S. Ceri, D. DeWitt, M. Franklin, H. Garcia-Molina, J. Gray, J. Held, J. Hellerstein, H. Jagadish, M. Lesk, D. Maier, J. Naughton, H. Pirahesh, M. Stonebraker, and J. Ullman. The asilomar report on database research, December 1998.

[8] 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.

[9] 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.

[10] Korn F., Sidiropoulos N., Faloutsos C., Siegel E., and Protopapas Z. Fast nearest neighbor search in medical image databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 215–226, Mumbai, India, 1996.

[11] 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.

[12] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 419–429, Minneapolis, May 1994.

[13] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30:170–231, 1998.

[14] A. Gersho. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, Boston, MA, 1992.

[15] 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.

[16] 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.

[17] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 562–573, September 1995.

[18] H. Karhunen. Uber lineare methoden in der wahrscheinlich-keitsrechnung. *Ann. Acad. Science Fenn*, 1947.

[19] N. Katayama and S. Satoh. The sr-tree: an index structure for high-dimensional nearest neighbor queries. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 369–380, Tucson, Arizona, May 1997.

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

[21] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84–95, January 1980.

[22] S. P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28:127–135, March 1982.

[23] M. Loeve. Fonctions aleatoires de seconde ordre. *Processus Stochastiques et Mouvement Brownien*, 1948.

[24] 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.

[25] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Math. Stat. and Prob*, volume 1, pages 281–196, 1967.

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

[27] 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.

[28] J. Nievergelt, H. Hinterberger, and K.C.Sevcik. The grid file: an adaptable, symmetric multikey file structure. *ACM Transactions on Database Systems 9*, 1:38–71, March 1984.

[29] 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.

[30] H. Samet. *The Design and Analysis of Spatial Structures.* Addison Wesley Publishing Company, Inc., Massachusetts, 1989.

[31] 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.

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