Simple and Compact Indexing for Efficient KNN Search in High Dimensional Feature Space

Zaher Al Aghbari
Department of Computer Science,
College of Sciences, University of Sharjah,
Sharjah, UAE
zaher@sharjah.ac.ae

Ayoub Al-Hamadi
Institute for Electronics, Signal Processing and Communications, IESK, University Magdeburg,
Magdeburg, Germany
Ayoub.Al-Hamady@ovgu.de

***Abstract*—In this paper, we propose a technique to find the exact KNN image objects to a given query object in high dimensional space. The proposed technique clusters the images using a self-organizing map algorithm and then it projects these clusters into points in a linear space based on their distances from a selected reference point. The projected points in a linear space are then organized in a simple, compact and yet fast index structure, called array-index. Unlike most indexes that support KNN search, the array-index requires a storage space that is linear in the number of projected points. The experiments show that the proposed technique is more efficient and robust to dimensionality as compared to other well known techniques due to its simplicity and compactness.**

Keywords-KNN search; image search; efficient indexing; dimensionality reduction; image clustering.

I. INTRODUCTION

Content-based retrieval of similar objects in a high dimensional feature space is important to many database applications. That is if an application is managing a database of *N* objects, given a query object *q*, the application should be able to return the *K*-Nearest Neighbors (KNN) objects in the database that are most similar to *q*. Finding KNN objects is one of the most expensive, but essential, operations in high-dimensional database applications.

In large databases, given a query *q*, finding the KNN answer set by a linear scan method is prohibitively expensive, particularly, if the database objects are high dimensional. Even with the existing indexing structures the response time of finding KNN answer set is equal to, or higher than, that of a linear scan due to a well-known phenomenon called *curse of dimensionality* [6][22]. Therefore, there is an essential need for efficient KNN search techniques. In this paper, we propose a technique that speeds up the KNN search of high-dimensional objects as compared to well-known methods, such as the R*-tree [3], SR-tree [14] and linear scan

Our approach is based on clustering the multi-dimensional objects (e.g., image data) using a self-organizing map (SOM) algorithm and then projecting the clusters into a linear (1-dimensional) distance space in which the projected clusters are ordered based on their similarity to a chosen reference point, *R*. These projected points in the linear space are inserted into the *array-index* [1]. Since the points in the arrary index are sorted, search for similar points to a given query can be achieved by a binary search whose complexity is $O(log_{N_C})$, where N_C is the number of indexed points. The search for KNN images start at the most similar cluster, called wining cluster (WC), to a given query and then proceeds to the left and right of the WC. The search continues till checking new clusters do not result in any new KNN images on both sides. This method retrieves exact KNN answer images to a given query image. More details of this technique can be found in [23].

Due to the small size of the index, our method requires a storage space that is linear in the number of generated clusters. As shown by the experiments, the proposed technique requires a search time that is much lower than that of the well known indexing structures R*-tree, SR-tree and linear scan, especially at high dimensions, which are bottlenecks for most KNN search algorithms.

The rest of this paper is organized as follows: Section 2 presents the related work. Section 3 discusses data clustering and the KNN search algorithm. The experiments are discussed in Section 4. Finally, we conclude the paper in Section 6.

II. RELATED WORK

Most works on finding the exact, or approximate, KNN points in a database to a given query point have provided good solutions for the low-dimensionality case, but for the high-dimensionality case there have been very little progress. To project high-dimensional objects to a lower dimensional space, some indexing structures such as the grid-file [18], K-D-tree [4] or LSD-tree [12] partition the data space into disjoint regions regardless of data clusters. On the other hand, data partitioning approaches like R*-tree[3], SR-tree [14] or X-tree [7] divide the data space according to the distribution of points in these data partition trees. Both the space and data partitioning approaches aim at speeding up the KNN search by pruning the irrelevant partitions to the given query. Generally, these approaches work well at low dimensions, but their performance degrades as the dimensionality increases [8].

To reduce the effect of high-dimensionality on KNN search, some methods apply "dimensionality reduction" techniques on the data and then insert the data into the indexing trees. The QBIC system [11] of IBM uses a bounding method to map the high-dimensional color histograms of images into points in a 3-dimensional space. Faloutsos et. al [9] used the Discrete Fourier Transform (DFT) to map subsequences of a time series into points in a 6-dimensional space.

Space-filling curve methods like the Z-order curve [20] or Hilbert curve [10] map the *d*-dimensional points into a one-dimensional space, i.e., $\Re^d \rightarrow \Re^1$. Generally, space-filling curve methods are suitable for approximate KNN search, but not exact KNN search. A filter based approach such as the VA-file [22] divides the data space into 2^b rectangular cells, where *b* denotes a user specified number of bits. The VA-file is an array of these compact approximations of points.

In summary, the current KNN search approaches, which are classified above, work well in low-dimensional spaces; however, their performance degrades in high-dimensional spaces. In high-dimensional spaces, there has not been a satisfying solution for the KNN search problem. Towards finding a satisfying solution, the proposed technique improves performance of the KNN search by clustering the image data and then projecting those clusters into points in a linear space. From these projected points, a simple index is built that supports efficient KNN search for images.

III. EFFICIENT KNN SEARCH

A. Image Representation

We use a wavelet transform to decompose an image into several frequency bands and then compute a feature vector from the coefficients of the low frequency band. Particularly, we use the Haar wavelets because it is the fastest to compute and simplest to implement as compared to the other wavelet bases. The significant features of the image are concentrated in the low frequency band, thus allowing the image coefficients to be truncated, keeping only a few coefficients that sufficiently represent the image. Moreover, such truncation process improves discrimination power because the process of matching feature vectors is focused on the significant features of images.

B. SOM-based Clustering

SOM is an unsupervised neural network that maps high-dimensional input data $\Re^n$ onto a usually two-dimensional output space while preserving the similarities between the data items [16]. The SOM consists of nodes (neurons) arranged in a two-dimensional grid.

Clustering qualities of the self-organizing maps (SOM) are comparable to other clustering methods, and additionally SOMs offer better representation of clustered data [19]. The SOMs are well known for their superiority in clusters' visualization of very large and high dimensional image databases due to the SOM property of preserving the clusters' structure within the data as well as inter-cluster similarity [13]. Due to this property, we chose to cluster the feature vectors of images by a SOM.

C. KNN Search Method

The proposed technique indexes the clusters of images rather than the images themselves. After clustering the images using the SOM method, the array-index [1] is constructed by the following steps:

(i) Select a *reference point* (*R*): For the image database that is used in our experiments, we found that selecting a center cluster as a reference cluster leads to a slightly faster search time than other selection schemes.

(ii) Compute the distance $D(R, m_i)$ between *R* and each cluster's representative codebook vector, m_i :

(iii) Project the cluster points into a linear space (array-index) based on the computed distances : The SOM clusters are projected into a linear space by inserting the computed distances $D(R,m_i)$ into the array-index. Along with each distance, a pointer to the best-matching-list of the corresponding cluster is set. The best-matching-list of cluster C_i is a list of the points that are associated with C_i and sorted according to their Euclidean distances from m_i. Hence, the array-index contains a list of clusters sorted based on their distances $D(R, m_i)$ and each cluster contains a pointer to a list of points sorted based on their distances $D(p_j, m_i)$. Also, for every C_i, we compute its radius r_i that reflect the size of C_i. The constructed array-index is very small in size since it only contains three fields (distance, pointer, and radius) for every generated cluster. Therefore, all, or most, of the array-index can be put in main memory; thus, eliminating, or minimizing, (depending on the number of clusters and size of main memory) the disk operations during the search.

D. KNN Search: Flow Control

The proposed KNN search algorithm starts by finding the WC, which is a cluster with the most similar codebook vector to q, by computing D(R, q) and projecting this distance on the array-index. The WC will be the left or right cluster to D(R, q), whichever closer to q, and in case of tie either choice will do. Then, as shown in Fig. 1, the algorithm continues the search for similar clusters on both the left and right directions of WC. During the KNN search we employ 4 conditions to control the flow and speed up the search process [1]:

- Cluster Pruning Condition to prune dissimilar clusters.
- Cluster Ending Condition to end the search as soon as all KNN points are retrieved and further checking of new clusters does not result in new KNN answer points.
- Point Pruning Condition to prune dissimilar points (images) in the currently visited cluster.
- Point Ending Condition to end the search for candidate points in the currently visited cluster if further checking will not result in new KNN candidate points.

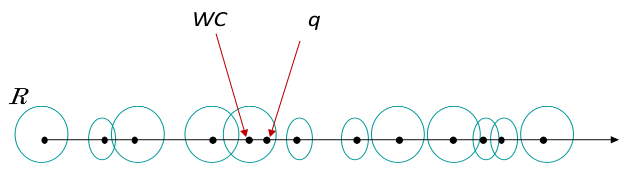

Figure 1. Searching to the left and right of the winner cluster (WC)

E. *KNNSearch Algorithm*

To find the exact KNN points to a given q, the *KNNsearch* algorithm first finds the *WC*, which is a cluster whose m_i is the most similar to q, within the array-index. Then, the *visitCluster()* function is called to visit the found *WC* and update *KNN_list* with the points associated with the *WC*. L_{ptr} and R_{ptr} are pointers used to traverse the array-index toward the left and the right of *WC*, respectively. Thus, after finding *WC*, the algorithm initializes L_{ptr} and R_{ptr} to point to *WC*.

The *traverseLeft()* and *traverseRight()* functions traverse to the left and right of *WC*, respectively, till one or both traversal paths are ended. If both traversal directions (left and right) are ended, the search stops and then the *KNNsearch* algorithm returns the *KNN_list*, which contains the exact KNN points to the given q. Otherwise, if both traversal direction (left and right) are not ended, the distances $D_L(q, m_{Lptr})$ and $D_R(q, m_{Rptr})$ are computed and the *visitCluster()* function is called to update the *KNN_list* with either the left, or right, current clusters (the one with the smaller distance from q), then the traversal in the opposite direction is blocked temporarily, so that the next iteration of the algorithm fetches the next partition in the direction of the *closer* cluster (the one with the smaller distance). When the left traversal is ended and the search is advancing only in the right direction, the *KNN_Search* algorithm updates the *KNN_list* with the next qualifying cluster on the right direction. Similarly, when the right traversal is ended and the search is advancing only in the left direction, the *KNN_Search* algorithm updates the *KNN_list* with the next qualifying cluster on the left direction. Finally, when the search stops, the *KNN_list* is returned with the exact KNN answer points.

The array-index speeds up the KNN search as follows:

i) Finding the *WC* takes $log_2\ N_c$ distance comparisons since the proposed technique uses a binary search as compared to a brute force method, which requires N_C distance comparisons.

ii) In the KNN search, the left and right traversal is ended as soon as the exact KNN answer points have been retrieved by the ending condition without the need to traverse the whole array-index. Thus, normally, only a small portion of the database is traversed to find the exact KNN answer points.

iii) Since the points associated with a cluster are sorted by their distances from the m_i, the ending condition ends the search at a certain point p beyond which there will be no KNN candidates.

IV. Experiments

We performed our experiments on collected images from publicly available image databases: H^2 Soft image databases and Stanford University database lab. The size of the collected image database is 80,000 artificial and natural images that span the following categories: landscapes, animals, buildings, people, plants, CG, etc. Where 50% of the images (40000 images) are used for training the SOM and the other 50% is used for testing. The image size is fixed at 128x128 pixels. Although the experiments are conducted on image vectors, the proposed algorithm can be applied on any high-dimensional vectors.

We extracted the color feature of each image using the Haar wavelets. The color space used in this paper is the YIQ since it exploits certain characteristics of the human eye, where Y represents the luminance information and I and Q represent the chrominance information [2][17]. The YIQ color feature for each image is represented by a d-dimensional vector. To demonstrate the efficiency of our KNN search algorithm, we generated 3 databases with different dimensionalities: 3, 12, and 48 dimensions by applying a 7-level, 6-level, and 5-level decomposition, respectively, of the Haar wavelet transform on the image data. To show the feasibility of our technique and its tolerance to high dimensions, we chose the 3, 12 and 48 dimensions since they represent a low, medium and high dimensionality, respectively, of the feature space.

From each database (3, 12, and 48 dimensions), we generated different data sets 1000, 5000, 10000, 20000, 30000, 40000 vectors in both the training and test databases. Each of these data sets was clustered by a SOM. The different dimensions were generated via applying Haar wavelets on the images.

To analyze the effect of dimensionality on the KNN search time, we measured the KNN search time of data sets of different sizes and different dimensionalities, as shown in Fig. 2. First, we randomly selected 10 query images from the database and issued a query to retrieve the KNN answer images (where K=1 and K=50) for each query image from each of the different data sets and different dimensionalities. The KNN search times shown in Fig. 2.left and Fig. 2.right are the average results of 10 queries. It is clear that the KNN search time increases as the dimensionality increases. That is because the volumes of clusters increase as the dimensionality increases causing a greater degree of overlap between clusters; thus, more clusters are visited during the search. Moreover, the KNN search time increases as the value of K increases. That is obvious because larger values of K lead to more clusters' visits; thus, larger percentage of the database is accessed to answer a query.

To prove the above reasoning, we measured the percentage of the non-empty clusters (associated with one or more images) accessed to retrieve the KNN images to each of the randomly selected queries at different dimensionalities and different values of *K* (see Fig. 3). Clearly, as the dimensionality increases and/or the value of *K* increases a bigger percentage of the database is accessed (more clusters are visited).

According to [15][22] and as verified by our experiments (see Fig. 4), most KNN search algorithms are outperformed by a linear scan method especially at high dimensions (≥ 12 dimensions) of real (non-uniformly distributed) databases, such as image databases. We compared the proposed technique with other well known methods (R*-tree, SR-tree and linear scan), which serve as a stick yard to evaluate the proposed technique, in terms of KNN search time versus the number of dimensions. The KNN search of the R*-tree and SR-tree methods is implemented based on the algorithm presented in [21]. Fig. 4 shows the comparison between the four methods for *K*=1 and *K*=50 using a test database of 40000 images. Notice that the scale of the *y*-axis of Fig. 10 is logarithmic.

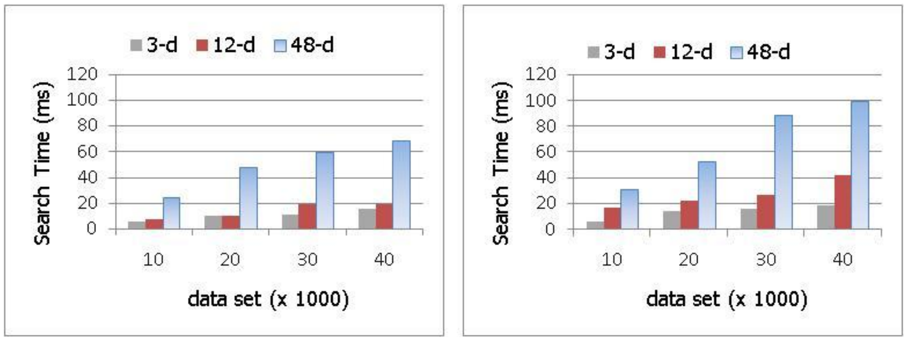

Figure 2. Average KNN time versus database size when (left) *K*=1 and (right) *K*=50.

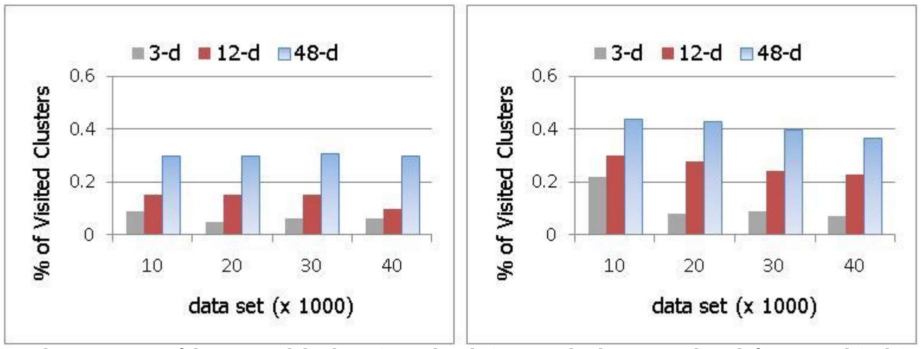

Figure 3. The percentage of the accessed database (visited nodes) versus database size when (left) *K*=1 and (right) *K*=50.

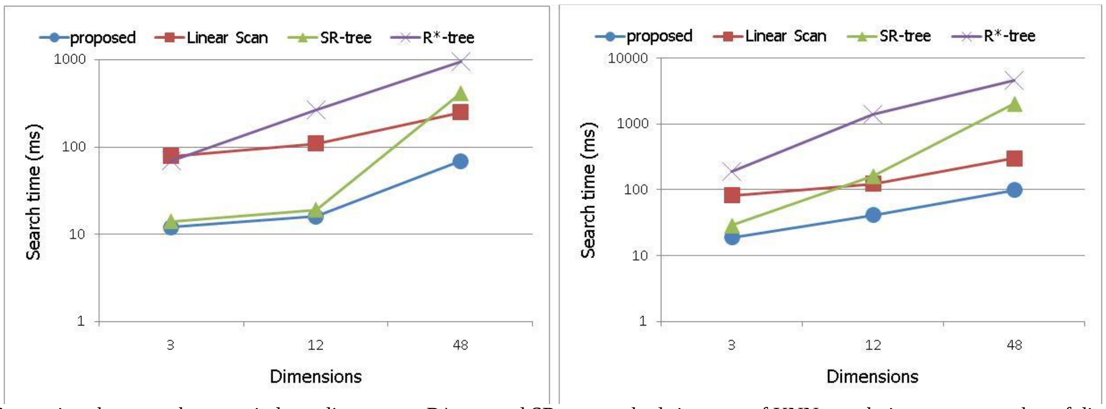

Figure 4. Comparison between the array-index, a linear scan, R*-tree and SR-tree methods in terms of KNN search time versus number of dimensions: (left) for *K*=1 and (right) for *K*=50.

Obviously, the proposed technique outperforms the R*-tree, SR-tree and linear scan methods at all dimensions and all variations of the value of *K*. At low dimensions and low values of *K* (dimensions = 3 and *K* = 1), the R*-tree and SR-tree generate their least disk operations; thus, they outperform the linear scan method. However, as the dimensionality increases and/or the value of *K* increases the disk operations of the R*-tree and SR-tree methods increase and hence they are outperformed by a linear scan. Due to the simple and compact structure of the array-index used by our technique, it generates no disk operations and thus it results in a significant performance gain over the R*-tree and SR-tree methods. Also, the proposed technique outperforms the linear scan method, on the average of the value of *K*, by 81% at 3 dimensions, 77% at 12 dimensions and 70% at 48 dimensions. Notice that the KNN search time of the linear scan methods increases linearly with the number of dimensions and the KNN search time of the R*-tree and SR-tree increase exponentially with the number of dimensions. On the other hand, the proposed technique shows more robustness to the number of dimensions since the KNN search time only increases slightly as the dimensionality increases.

V. CONCLUSION

We presented a simple and efficient technique to retrieve exact KNN points for a given query. The proposed technique is based on clustering the data and then indexing the clusters' representative vectors instead of the data points themselves; thus, reducing the size the index structure. Although, in this paper, we used an image database and a SOM algorithm as examples of a multi-dimensional data and a clustering technique, respectively, the technique can be adapted to any multi-dimensional data and any clustering method. Even though the paper only presents the KNN search algorithm, the range search can be easily implemented. Given q a range query ρ, the search algorithm first finds the WC, next it retrieves all clusters that overlap with this range on both sides of the WC as a candidate set. Finally, the candidate set is refined by retrieving only the points p_i that satisfy the range condition $|D(q, p_i)| \leq \rho$. The projection of clusters into a linear space and the compactness of the index are the main reasons behind the fast KNN search time as compared to other known methods. The experiments show that the proposed technique is faster and more robust to dimensionality as compared to other well known techniques due to its simplicity and compactness.

REFERENCES

[1] Z. Al Aghbari,. "Array-index: A Plug&Search K Nearest Neighbors Methods for High-Dimensional Data," Journal of Data & Knowledge Engineering, Vol 52, no.3, pp 333-352, Mar. 2005.

[2] Z. Aghbari, K. Kaneko, and A. Makinouchi, "New Indexing Method for Content-Based Video Retrieval and Clustering for MPEG Video Database," International Symposium on Digital Media Information Base (DMIB'97), p Nov.1997, pp.140-149.

[3] N.Beckmann, H.Kriegel, R.Schneider, and B.Seeger, "R*-tree: An Efficient and Robust Access Method for Points and Rectangles," ACM SIGMOD, May 1990, pp. 322-331.

[4] J.L.Bentley, "Multidimensional Binary Search Trees in Database Applications," IEEE Trans. on Software Engineering, SE- vol. 5, no. 4, July 1979, pp.333-340.

[5] S.Berchtold, C.Bohm, D.Keim, and H.P.Kriegel, "A Cost Model for Nearest Neighbor Search in High-Dimensional Data Spaces," ACM SIGACT-SIGMOD-SIGART, 1997.

[6] S.Berchtold, B.Ertl, D.Keim, H.P.Kriegel, and T.Seidl, "Fast Nearest Neighbor Search in High-Dimensional Space," Int`l conf. on Data Eng. (ICDE), 1998.

[7] S.Berchtold, D.Keim, and H.P.Kriegel, "The X-tree: An Index Structure for High-Dimensional Data," VLDB 1996.

[8] C.Faloutsos, "Searching Multimedia Databases By Content," Kluwer Academic Publishers, Boston, USA, 1996.

[9] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," ACM SIGMOD, 1994.

[10] C.Faloutsos and S.Roseman, "Fractals for Secondaru Key Retrieval," 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS), March 1989.

[11] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D, Lee, D. Perkovic, D. Steele, and P. Yanker, "Query by Image and Video Content: The QBIC System," IEEE, Sept. 1995.

[12] A.Henrich and H-W.Six, P.Widmayer, "The LSD tree: Spatial Access to multidimensional Point and Non-Point Objects," Proc. of 15th VLDB, 1989.

[13] J.Himberg, "A SOM based cluster visualization and its application for false coloring," International Joint Conf. in Neural Networks, 2000.

[14] N.Katayama and S.Satoh, "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries," ACM SIGMOD, May 1997.

[15] J.M.Kleinberg, "Two Algorithms for Nearest Neighbor Search in High Dimensions," 29th ACM Symposium on Theory of Computing, 1997.

[16] T.Kohonen, "Self-Organizing Maps," Springer-Verlag, 1997. 2nd extended edition.

[17] M. Kubo, Z. Aghbari, K-S Oh, and A. Makinouchi, "Content-Based Image Retrieval Technique using Wavelet-Based Shift and Brightness Invariant Edge Feature," International Journal on Wavelets, Multiresolution and Information Processing (IJWMIP), vol.1, no.2, 2003, pp. 163-178.

[18] J.Nievergelt, H.Hinterberger, and K.Sevcik, "The grid file: An Adaptable Symmetric Multikey File Stucture," ACM Transaction on Database Systems, vol. 9, no. 1, 1984, pp. 38-71.

[19] K. S. Oh, Z. Aghbari, and P-K. Kim, "Fast k-NN Image Search with Self-Organizing Maps," CIVR 2002.

[20] J.Orenstein, "Spatial Query Processing in an Object-Oriented Database System," Proc. ACM SIGMOD, May 1986.

[21] N.Roussopoulos, S.Kelley, and F.Vincent, "Nearest Neighbor Queries," ACM SIGMOD, 1995.

[22] R.Weber, H-J.Schek, and S.Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," Proc. of the 24th VLDB, USA, 1998.

[23] Z. Al Aghbari and A. Al-Hamadi, "Efficient KNN Search by Linear Projection of Image Clusters," Wiley International Journal of Intelligent Systems, Vol. 26, no. 9, 2011, pp. 844-865.