One-Dimensional Index for Nearest Neighbor Search *

Theen-Theen Tan[1], Larry Davis[1], and Ramki Thurimella[2]

[1] Dept. of Computer Science, University of Maryland, College Park, MD 20742, USA
[2] Dept. of Math and Computer Sciencee, University of Denver, Denver, CO 80208, USA

Abstract. We describe a study of a multidimensional indexing scheme to solve the nearest neighbor problem based on mapping the high dimensional space onto one dimension. The idea can be seen as linearizing the data points in the multidimensional space along a curve. We propose using a genetic algorithm to find a good one-dimensional mapping for a given data set. Second, we look at using two non-correlated curves for indexing, and examine the accuracy and efficiency trade-off using the proposed scheme as opposed to nearest neighbor search by scanning the space linearly. Experiments are conducted on a face image recognition application.

1 Introduction

One of the most frequently performed tasks in multimedia, data mining, and information retrieval systems is to find objects similar to query objects. The data in these applications is usually represented by a set of features that is an abstraction of the data. A data point can be seen as a vector in the feature space. The similarity measure can be regarded as the distance between objects within the feature space. A search for similar objects is then a nearest neighbor search in that space .

The simplest approach to the nearest neighbor problem is to compare a query against the entire database linearly to find the solution. This method is costly as the search time increases proportionally with the number of elements in the data space. Various solutions have been proposed to make the nearest neighbor search more efficient. Among them are various hierarchical indexing techniques such as KD trees and Quadtrees [12], R trees [6], its variants, and TV-trees [8]. These indexing methods work well for low-dimensional spaces, but their performances degrade as the number of dimensions increases. This problem is known as the 'dimensionalilty curse'. As the dimension of the feature space grows, the performances of these methods can become worse than a linear scan through the database [15]. The computational complexity for a nearest neighbor

* The support of this research by the Department of Defense under contract MDA 9049-6C-1250 is gratefully acknowledged.

search in these indexing methods is proportional to 2^d. The number of dimension of feature vectors can be high in certain applications. For example, in typical image database search, the feature space often is $256 * 3$ dimensions, using color histogram as features. Therefore, various ways have been proposed to reduce dimensionality of d-dimensional feature space to k-dimensions ($k < d$) so that the mentioned multidimensional indexing schemes can be effectively used. The *Karhunen-Loeve* transform [5], multidimensional scaling [7], and FastMap [4] are exa mples of such feature dimensionality reduction schemes.

We are interested in studying indexing methods that avoid the 'dimensionality curse' problem. Some recent research in solving the nearest neighbor problem in the high dimensions feature space include VA-files [15] and the Pyramid technique [1]. VA-files compute an approximated distance for each data point from the query point using information in a signature file. The Pyramid technique partitions the d-dimensional space such that it can be accessed in "stages". We present an indexing scheme for multidimensional data based on mapping the data points onto one-dimensional curves. Research closely related to our work are [3] and [13] which use various space-filling curves as indexing schemes.

Many space-filling curves have been proposed in the literature. Widely known examples are *z-ordering* [11], Hilbert [2], Peano, and Gray. All the discrete mappings of these curves can be efficiently computed by bit manipulation. For examples, *z-ordering* is based on interleaving bits from the coefficients. A property of any space-filling curve is that neighboring points along the curve are also close in the original space, but not vice-versa. The result is that certain neighboring pairs in the original space are mapped far apart along the curve.

The attractiveness of having a good clustering of multidimensional points along a one-dimensional mapping is that we can minimize the number of disk access in a range query. With data spread along only one dimension, we can use range trees based on a binary tree or a B-tree as an indexing structure. A nearest neighbor search in this scheme involves a range search along the curve. We label the nearest neighbor in the original feature space as the NN_d. The search radius of the range search needed to find the NN_d can be large due to the discrepency of nearest neighbor distances along the curve as discussed in the previous paragraph. To solve this problem, previous research on using space-filling curves as indexing methods have used multiple curves hoping that the curves will act in collaboration and compensate each other. In [3], two standard space filling curve, Hilbert and Peano, are used. In [13], a Hilbert curve and 40 space filling curves generated by translating and permuting the bit representation of the original feature vector are used. [13] reported that the number of curves needed to maintain the same level of accuracy increases with the number of dimensions for various applications. However, it is unclear what the complexity relationship between d and the number of curves needed is.

Our objective is to find a good one-dimensional mapping for a specific data set to solve the nearest neighbor problem. The one-dimensional mapping that we

search for can be constructed by selecting and permuting the bit representations of the original features. We call this resulting bit representation the *SuperKey*. A binary range tree is used as our index structure. A nearest neighbor search for a query involves (Figure 1) :

1. Find the nearest *SuperKey* neighbor (NN_s) in *log n* time using the binary tree created on the *SuperKey*.
2. Using the original *d*-dimensional features, conduct a linear nearest neighbor search on a fixed interval of leaf nodes centered at the NN_s of the binary tree to find the true nearest neighbor,NN_d.

Our goal is to find the *SuperKey* that minimizes the radius of the search interval required in (2). We use a genetic algorithm to search for a mapping that minimizes this radius. We also propose a method to identify additional one-dimensional mappings which complement the previous mappings. Details of our algorithm are described in the next section.

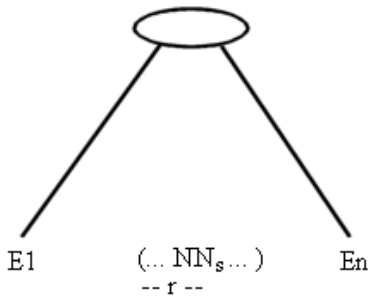

Fig. 1. Visualization of our algorithm. The search on *SuperKey* index represented by a binary tree will bring us to the nearest neighbor in the *SuperKey* space (NN_s). A linear scan is done in a search interval to find the true nearest neigbor (NN_d). We seek to minimize the search radius

2 Optimal Mapping Search

2.1 Genetic Algorithm

Genetic algorithms [9] are known for their capability of searching a high dimensional space for a "close to optimal" solution. Similar to simulated annealing, they approximate the global optimal by iteratively varying the current solutions through crossover and mutation operators until a good solution is found. The difference is that genetic algorithm optimizes by searching various locations in the solution space through a population of possible solutions. The new locations

to be searched are produced by mating of individuals in the previous generations. Crossover and mutations are the mating operators. The crossover operators find various localities to search, and the mutation operator perturbs search out of their local minima. The genetic algorithms will evolve a number of generations searching for an optimal solution.

For a specific problem to be solved by a genetic algorithm, the essential components that must be defined are the encoding of the genomes, the crossover and mutation operators, and the objective function. In our application, a genome is a vector of pairs. The first element in the pair is a feature in the original vector, and the second element specifies the number of significant bits to extract from the bit string of the particular feature. We use an *EvenOdd* crossover operator in which one child inherits the odd genes of the parents and the other child inherits the even genes of the parents. We studied two mutation operations. The first is to increment or decrement the number of most significant bits, and the second is to randomly swap places with another gene in the genome. The crossover operator causes most of the changes in the arrangement of the genomes. Notice that the *EvenOdd* crossover may cause a child to contain duplicate genes. Assuming that genomes containing duplicate genes will hold less information for matching, and therefore die out eventually, we did not modify the *EvenOdd* operator to eliminate duplicate genes. Indeed, all of the genomes in the final populations throughout our experiments contain no duplicate genes. The objective function is to minimize the radius we need to search to find the true nearest neighbor.

We used a Steady State genetic algorithm, in which a certain percentage of the previous population is used as the initial population for the next generation of evolution. There are other options for a genetic algorithm, such as selection criteria (i.e. which genomes should be selected to mate), replacement strategy (which individuals should be replaced), and scaling (converts objective score to a fitness score to be used during selection). We use the simplest of these choices which are Roulette wheel selection, replace the weakest individuals from the population, and linear scaling. Refer to [9] for additional details on genetic algorithms.

2.2 Indexing and Searching

From the best solution found by the genetic algorithm, we construct the *SuperKey* according to the order of the features and number of significant bits,b, indicated by the gene pairs. The data structure of a *SuperKey* is an array of bit strings simulating a very long integer of b * number of coefficient bits. The training elements are then sorted according the values of the *SuperKey*. A query then is a binary search with the *SuperKey* of the test element against this sorted list, followed by a linear search in a fixed size radius as described in Section 1.

We mentioned previously that using more curves would improve the accuracy of the nearest neighbor queries. We seek additional curves that will improve on

the results of using one curve. The way we find such curve is to train our second curve on that subset of training cases that do not perform well on the testing using the first curve. One can visualize this method as finding additional curves that connect nearest neighbor pairs which were places far apart in the first curve. A search with multiple curves would then involve indexing the query element along each curve and scanning through the fixed search radius to find the NN_d on each of the curves. Of course, if we use k curves, the required radius on each curve must be at least k times small than the search radius needed for only one curve to find the NN_d. We show experimental results using only two curves.

2.3 Complexity

The majority of time spent in our algorithm is in the genetic algorithm training stage. The construction of the *SuperKey* is constant time with respect to the number of elements in the database, linear with respect to the number of dimensions. Any ordinary sort algorithm that runs in *O(n logn)* can be used. An insertion or a deletion into the index for each curve takes *log n* time. The query time is merely a *log n* time binary search on each index, plus a linear scan through a small subset of elements contained within a radius about the element being accessed with the binary search.

3 Experiment

The data we use for the experiment is taken from the Feret face database [1]. The database contains faces of 431 people, with two images of each person. We use a jack-knife approach in our experiment. Training is done on a subset of the database, containing one image of each individuals and tested on the other and vice versa. A popular feature extraction method for face recognition applications is the principal component analysis, also known as Eigenfaces[14]. It is shown that using this method, a recognition accuracy of more than 90% can be achieved. The pixel values of each image are treated as a long vector. The eigen vectors of the training set are calculated, and each training image is then projected onto the eigen vectors to obtain a vector of projection coefficients. In recognition, each test image is projected onto the trained eigen space. A brute force linear search is usually used to find the trained projection vector that is most similar to the test vector. We use the first 180 eigen coefficients as our feature vector. We translated the coefficient space by a constant so that all the coefficients are positive. This step is necessary to avoid the problem of having to deal with two's complement's bit representation, which will affect the search for our *SuperKey* mapping.

[1] We thank Saad Sirohey, Arvindh Krishnaswamy, and Wenyi Zhao from U. of Maryland at College Park Center For Automation Research for providing us the data set

As input to the genetic algorithm, we need to create an initial population. Starting with a genome with 180 features arranged in ascending order with five significant bits from each feature, we randomly swap places between 30 of gene pairs. Each initial population contains 100 individuals. The crossover probability we used is 0.95, mutation probability is 0.01, and we carry over half of the population to the next generation. For each experiment, we randomly choose fifty images from the training set to facilitate the evaluation of the objective function during the evolution of the genetic algorithm. For our solution using multiple curves, we train for the first curve as described above. We tested the first curve on the test set, and used fifty training cases which do not perform well as the training set for the second curve. In all experiments, the genetic algorithm converges to a solution after evolving for about 30 generations.

4 Results

As a measure of accuracy, we calculated the probability, p, of correct answers (NN_d) found within a search radius, r. The number of elements to be searched is $2*r$, as we have to search the elements above and below the ordering along the curve about an element. We let c be the number of curve used in the indexing and e be the number of elements that we have to compare against to find the nearest neighbor ($e = \sum_{i=1}^{n} c_i * r_i$).

First, we present results using only one curve. To ensure that the result from the genetic algorithm is not dependent on the initial population, we ran the genetic algorithm on three different initial populations. We used *EvenOdd* crossover and tried on both mutation operators. The solutions for the first mutation operator generally perform better than that of the second mutation operator. So, we only report results of experiments using the first mutation operator. Notice that difference in the first 14 bits of the *SuperKey* is enough to discriminate over 10,000 elements. Since the best individuals from the populations differ only starting from the ninth coefficient and our database is small, we get the same ordering for the elements in our database from the best individuals from the three initial populations. The best individuals we obtained from training on all the three populations indicate using the first five significant bits in the coefficient arrangement of increasing order, starting with the first. This is equivalent to row major order. We compare the resulting ordering indicated by the *SuperKey* against z-ordering (interleaving bits from coefficients) of the space. Figure 2 shows a plot of the accuracy against the search radius for all 860 test images. The thin line shows the result for *z-ordering*. The thick line shows the result from the ordering indicated by the genetic algorithm. The performance for the two diverges as the accuracy increases beyond 70%. To achieve 90% correct query, we need to search a radius of 174 on the *z-ordering*, but only 89 for the ordering we found. Next, we report on improvements using multiple curves.

The mappings produced for the second curve are different for the all three initial populations. They do not correspond to any standard ordering at a glance.

Table 1 shows the average accuracy across the three initial populations for a fixed radius being searched. *Curve 1* is the curve found form the initial round of training. *Curve 2* is the additional curve found from training the elements with poor search result in the first curve. The radius needed using multiple curves is the minimum of the radii from the two curves. By searching the minimum radius on both curves, we are guaranteed to find the correct nearest neighbor. Recall that when using multiple curves, we have to search for the nearest neighbor in both curves. Therefore, we should compare the results for multiple curves with the results from a single curve using half the search radius. As expected, the accuracy for *Curve 1* is better than *Curve 2* since additional curves are seen as complements to previous curves. Comparing the accuracy of using two curves for $r = 10$ with accuracy of using only *Curve 1* for $r = 20$, we get 15.27% improvement. Comparing the accuracy of using two curves for $r = 20$ with accuracy of using only *Curve 1* for $r = 40$, we get 13.98% improvement. To get an accuracy of 90%, we have an average radius of searching on $r = 25$ for using two curves from the three populations ($r_1 = 25, r_2 = 28, r_3 = 23$). This means we compare a total of 100 elements on both curves, about 23% of all the possible answers. This is about half the number of elements that must be compared to using only one curve (*Curve 1*). It is likely that as we increase the number of curves according to the suggested procedure that we will be able to further improve on the result.

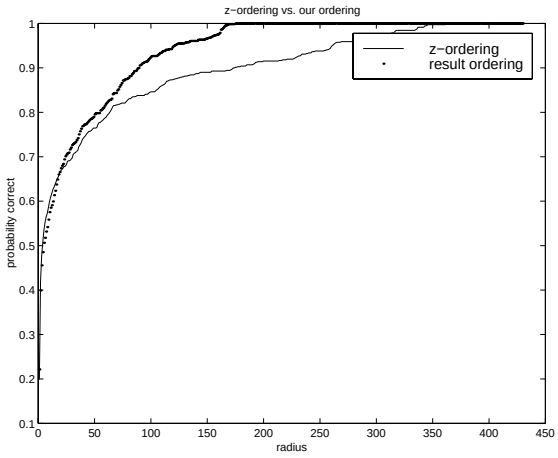

Fig. 2. Plot of accuracy vs. search radius comparing *z-ordering* vs. the genetic algorithm ordering

Table 1. Accuracy over fixed radii averaged across the three runs using different initial populations for using *Curve 1*, *Curve 2*, and both curves

	r = 10	r = 20	r = 40
Both	0.78	0.87	0.94
Curve 1	0.58	0.67	0.76
Curve 2	0.56	0.65	0.74

5 Conclusion

We proposed a solution to the nearest neighbor problem in high dimensional space. The optimal solution we propose is data dependent. Initial results are promising in that a data-dependent optimal solution works better than certain standard space-filling curve mapping. Our proposal to use multiple non-correlated curves also shows better results than using only one single curve. To complete the study, we would like to experiment with k-nearest neighbor search, use a larger database and data of different nature, and compare against other standard space-filling curve. It is an open problem whether a set of optimal one-dimensional curves exists for arbitrary data sets.

References

1. S. Berchtold, C. Kriegel, H. Hriegel. The pyramid-tree: Breaking the curse of dimensionality. Proc. ACM SIGMOD, 142-153, 1998
2. A. Butz, "Convergence with Hilber's space filling curve", J. Computer and Systems Science, **vol. 3**, 128-146, May 1969.
3. S. Craver, B. Yeo, M. Yeung. Multi-Linearization Data Structure for Image Browsing. SPIE Conference on Storage and Retrieval for Image and Video Databases VII, 155-166, January 1999.
4. C. Faloutsos and K. (David) Lin FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. ACM SIGMOD, May 1995, San Jose, CA, pp. 163-174.
5. K. Fukunaga. Introduction to Statistical pattern Recognition. Academic Press, 1990. 2nd. Edition.
6. A. Guttman. R-trees: a dynamic index structure for spatial searching. Proc. ACM SIGMOD, 47-57, June 1984.
7. J. B. Kruskal, M. Wish. Multidimensional scaling. SAGE publications, Beverly Hills, 1978.
8. K. Lin, H. Jagadish and C. Faloutsos. The TV-tree - An Index Structure for High-dimensional Data. VLDB Journal, 3, 517-542, Oct. 1994.
9. M. Mitchell. An Introduction to Genetic Algorithms. MIT Press, Cambridge, MA, 1996.
10. B. Moon, H.V. Jagadish, Christos Faloutsos and Joel H. Saltz, under revision for publication in IEEE Transactions on Knowledge and Data Engineering
11. J. Orestein. Spatial query processing in an object-oriented database system. Proc. ACM SIGMOD, 326-336, May 1986.

12. H. Samet. The Design and Analysis of Spatial Data Structure. Addison Wesley, 1989.
13. J. Shepherd, X. Zhu, N. Megiddo. A Fast Indexing Method for Multidimensional Nearest Neighbor Search. SPIE Conference on Storage and Retrieval for Image and Video Databases VII, 350-355, January 1999.
14. M. Turk, A. Pentland. Eigenfaces for Recognition. Technical Report #154, MIT, 1991.
15. R. Weber, H. Schek, S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. Proc. Of VLDB, August 1998.