Comparison of Two Fast Nearest-Neighbour Search Methods in High-Dimensional Large-Sized Databases*

Javier Cano, Juan-Carlos Pérez-Cortés, and Ismael Salvador

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia,
Camino de Vera, s/n 46071 Valencia, Spain
{jcano,jcperez,issalig}@iti.upv.es

Abstract. In this paper we show the results of a performance comparison between two Nearest Neighbour Search Methods: one, proposed by Arya & Mount, is based on a $kd-$tree data structure and a Branch and Bound approximate search algorithm [1], and the other is a search method based on dimensionality projections, presented by Nene & Nayar in [5]. A number of experiments have been carried out in order to find the best choice to work with high dimensional points and large data sets.

1 Introduction

Nearest Neighbour classifiers are widely used in the field of Pattern Recognition. In this and other areas, fast algorithms are needed that efficiently solve the *nearest neighbour search* problem, defined as follows: given a set P of points in a high dimensional space, construct a data structure that given any query point q finds the point in P closest to q. This definition involves the notion of a distance between two points. In our case the Euclidean distance has been used.

This and other problems of the same class (closest pair, diameter, minimum spanning tree, etc.) have been investigated in the field of computational geometry and many efficient solutions have been found that work reasonably well in low-dimensionality spaces. As the dimension scales up, however, many methods experiment serious performance degradation, with space and/or time requirements tending to grow very fast in most cases.

One way to alleviate this problem is to allow for approximate solutions rather than always forcing the search of the exact ones. In fact, some practical applications show no significant loss of precision when approximate nearest neighbours instead of the exact ones are used.

Several fast approximate search algorithms have been proposed in the literature. In this work we have focused on two of them. The first one is based on a well known data struture: the $kd-$tree, and an efficient approximate search algorithm that uses incremental distance calculations and a bound threshold in the search to save computing time. Previous work with this method, as in [3],

* Work partially supported by Spanish CICYT under grant TIC 2003-08496-C04-02.

A. Fred et al. (Eds.): SSPR&SPR 2004, LNCS 3138, pp. 868–875, 2004.

[2], shows that it is a valid option to cope with the problem of nearest neighbour in high-dimensional data and large-size databases.

The second solution achieves good average search times thanks to the simplicity of the method and the careful design of a precomputed data structure composed by a set of ordered lists of pointers. Then, when a search has to be performed, an optimized algorithm proceeds with an iterative list trimming process, which finally produces a short list of candidate points. A distance computation is then performed for each of this points and the closest one is the nearest neighbour of the query point [5].

In this work, a comparison of the two methods is carried out. A direct exhaustive search method has also been tested as a reference baseline providing us with an upper bound of the average time required to perform a search.

2 Corpora

Both real and synthetic databases have been used in the experiments presented. For the real data, the well-known OCR NIST databases have been chosen. Specifically, the upper-letter set of the NIST Special Database 3, which is composed of 44.951 images, has been used for training, and the NIST Special Database 7, composed of 11.941 images, has been used for test.

Each original 128x128 pixel binary image has been cropped and rescaled to a 9x12 pixel grey-level image. Then, a 108 component vector is built by simply concatenating the rows of an image. Finally, Principal Component Analysis (or Karhunen-Loeve Transform) is performed to reduce the dimensionality to 30.

Two different kinds of random distributions have been used to generate the synthetic data sets: normal, with $\mu = 0.0$ and $\sigma = 1.0$, and uniform, in the unit hypercube. In both cases, a set of 50.000 points has been generated for training and 10.000 points for test. The points, as before, are always represented by 30-dimensional vectors.

3 Search Methods

As discussed before, the problem of searching the nearest neighbour can be trivially solved by an exhaustive search in a simple list. However, the potentially huge size of the prototype set and the high dimensionality of data makes it extremely inefficient and, therefore, inappropiate for real tasks. Instead of this naive solution, several sophisticated methods have been developed in order to achieve approximate searches with very good average search times at the expense of a little increase of the error rate. On the next sections two of these methods are reviewed and some interesting propierties are remarked.

3.1 Kd-Tree + ANN

The $kd-$tree is a widely used data structure. It is a binary tree where each node represents a region in a $k-$dimensional space. Each internal node also defines a

hyper-plane (a linear subspace of dimension $k-1$) dividing the region into two disjoint sub-regions, each inherited by one of its sons. Most of the trees used in the context of our problem divide the regions accordingly to the points that lay in them. This way, the hierarchical partition of the space can either be carried out to the last consequences to obtain, in the leaves, regions with a single point in them, or can be halted in a previous level so as each leaf node holds b points in its region. This number of points, b, is commonly referred as the *bucket size*.

The search of the nearest neighbor of a test point in a $kd-$tree is performed starting from the root, which represents the whole space, and choosing at each node the sub-tree that represents the region of the space containing the test point. When a leaf is reached, an exhaustive search of the b prototypes residing in the associated region is performed. But the process is not complete at this point, since it is possible that among the regions defined by the initial partition, the one containing the test point does not contain the nearest prototype. If this happens, the algorithm backtracks as many times as necessary until it is sure to have checked all the regions that can hold a prototype nearer to the test point than the nearest one in the original region. The resulting procedure can be seen as a Branch-and-Bound algorithm.

As stated before, fast approximate nearest neighbors search algorithms allow for efficient classification without losing significant precision. The notion of approximate search can be viewed as a slightly modified search where instead of reporting a point p closest to q, the algorithm is allowed to report the first point found within a distance $(1+\epsilon)$ times the distance from q to p.

An algorithm based on these concepts was proposed by Arya & Mount [1]. The algorithm, referred in the article as "Standard k-d Tree Search with Incremental Distance Calculation", works as follows. At each leaf node visited they compute the squared distance between the query point and the data point in the bucket and update the nearest neighbour if this is the closest point seen so far. At each internal node visited they first recursively search the subtree whose corresponding rectangle is closer to the query point. Later, they search the farther subtree if the distance betweeen the query point and the closest point visited so far exceeds the distance between the query point and the corresponding rectangle.

We have used it on conventional $kd-$trees in the experiments. We will refer to this algorithm as ANN (Approximate Nearest Neighbour).

3.2 Projection Based Search

This method is inspired on the original projection search paradigm [4]. However, when dealing with high dimensionalities, a naive version of the basic projection concept is not efficient enough in most cases.

The refined projection search method proposed by Nene & Nayar [5] can be summarized as follows: first, given a query point q, those prototypes in the first dimension which fall into the range $[q_0-\epsilon, q_0+\epsilon]$ are selected to constitute the first *candidate list*. Then, for the other dimensions, the bounds $[q_d-\epsilon, q_d+\epsilon]$ are computed and the *candidate list* is trimmed according to them. After the

trimming process has been performed through all the dimensions, only a short list of prototypes which fall into the hyperrectangle defined by q and ϵ remains in the final *candidate list*. Finally, the distances from the query point to the prototypes included in the final *candidate list* are exhaustively computed, and the nearest prototype are returned.

A description of this method and some guidelines for an efficient implementation can be found in [5], where taking advantage of a precomputed data structure, a set of ordered lists, it is possible to perform a nearest neighbour search by only computing a small number of binary searches and integer comparisons.

The main problem with this method is its high sensivity to an inadequate estimation of ϵ. An excessively small value leads to an empty final candidate list. A large value of ϵ will produce a large candidate list and, consequently, a performance degradation.

Another important problem of the technique is related to the metric used in the candidate list trimming: the algorithm discards points that fall out of a threshold ϵ defined by an L_∞ distance, but we are searching for the L_2 nearest neighbour, therefore, if the threshold used is too small, the nearest neighbour can be trimmed and the final candidate list not be empty. Thus, when the Ecuclidean Distance is used as the metric, the method is in fact an approximate nearest neighbour search algorithm.

Moreover, the probability of finding other prototypes in an hyperrectangle that does not include the true nearest neighbour (the volume of the hyperrectanlge that is outside an inscribed hypersphere) grows with the space dimensionality.

A further degree of flexibility is possible if we allow the size of the intervals used to build and trim the lists to vary for the different coordinates, or even adaptively adjust for each component of each query point. This means that we can use a different ϵ_d for each axis, and change it, if desired, accordingly to the position of the query point in the space.

In order to evaluate the method, a baseline version has been implemented. We have assumed no "a priori" knowledge of the data distribution. Then, at the design phase the variance of every dimension d (σ_d^2) in the prototype set is computed and stored into a vector. Then, in the search phase, the interval size ϵ_d for each dimension is simply chosen as $\epsilon_d = C * \sigma_d^2$.

On the other end of the performance spectrum, an *optimistic, ideal version* of the algorithm has also been implemented in order to find the best search times that could be reached by the projections based search method. The idea consists on providing the algorithm with the ideal parameter setting for each single search. This has been accomplished by precomputing the difference from the query point to its true nearest neighbour in each coordinate. The ϵ_d associated to the corresponding projection is then set to that value (potentially different for each query point and each dimension). This finally produces, for each search, a candidate list with the minimum possible size including the *nearest neighbour*. No other combination of parameters can yield a faster search. Of course, in a real

application setting, any conceivable (efficient) method to estimate the different ϵ_d will probably result in significantly longer running times.

4 Experiments

A first experiment was aimed at defining a lower bound in the average time required by the Nene & Nayar approximation. Making use of the precomputed ϵ_d for each test point, the same precision of an exhaustive search was achieved with a 27.58 times faster average search time.

The conditions of all the experiments were: AMD Athlon Processor at 1200Mhz, 640 Mb of main memory, Linux operating system kernel version 2.4.22 and gcc C compiler, version 3.3.2.

Table 1. Comparison of average execution times with real data (NIST SD3-SD7). An absolute ideal lower bound is shown for the projections (Nene & Nayar) approximation.

Search Method	Avg. Time	Error Rate
Exahustive	25.93 ms.	12.13%
***kd*-tree+ANN** (ϵ=0.0)	9.45 ms.	12.13%
Projections (ϵ_d precomputed)	0.94 ms.	12.13%
***kd*-tree+ANN** (ϵ=2.0)	0.45 ms.	12.32%

As can be seen in Table 1, the projections method does not improve on the speed up of the *kd*-tree+ANN if a little precision loss is allowed. Taking into account that precomputed ϵ_d (the absolute best possible value of that parameter for each dimension) have been used and that they are a key factor in the performance of the method, it is clear that unless we have a way to precisely estimate these parameters, the method will not work better than the *kd*-tree+ANN.

An experiment has been performed with this approximate method using the simple parameter specification technique described in the previous section and modifying the constant C, which took values in $\{0.5, 1.0, 1.1, 1.2, 1.5, 1.8\}$

In order to compare the performance of the two algorithms, the results of the *kd*-tree+ANN method, with ϵ taking values in $\{0, 1, 2, 8, 64, 2048\}$, are shown in Figure 1. It can be seen there that for all tested values, *kd*-tree+ANN always outperforms Projections. The best possible performance of the Nene & Nayar method achieved with the precomputed ϵ_d for each test point is also plotted in the graph as a small cross. Its position, very near the *kd*-tree+ANN curve suggests that it is very unlikely that a real implementation of the Nene & Nayar method can be superior in practice to a *kd*-tree for the task examined.

Another set of experiments have been performed to compare the algorithms in a more controlled, synthetic, setting. In this case, instead of the classification error in a Pattern Recognition task, the search performance has been directly measured, therefore no classification labels have been used for the data. The

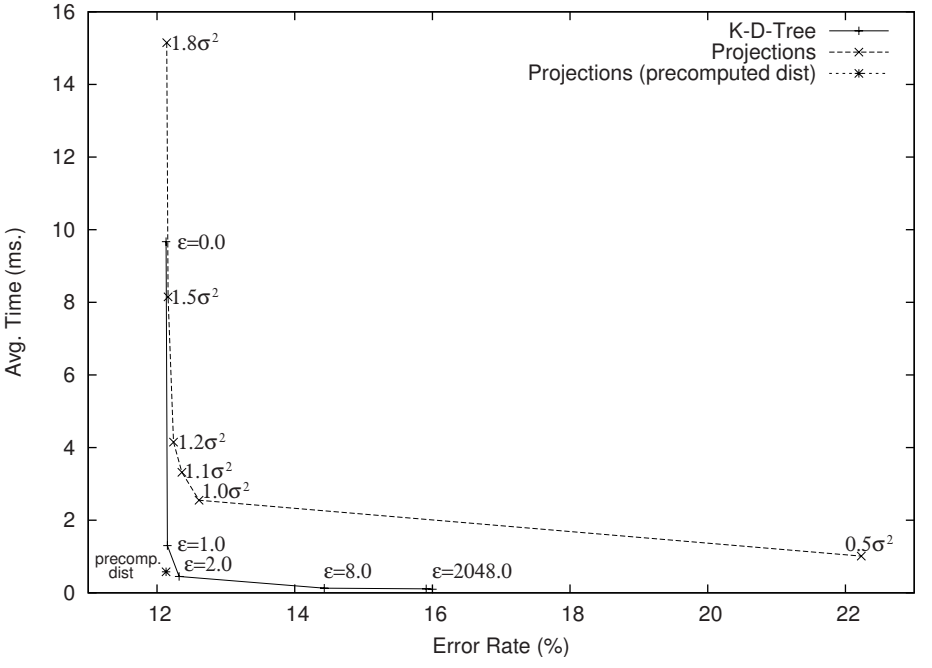

Fig. 1. *kd*-tree+ANN vs. Projections with real data (NIST SD3-SD7). A baseline version of the projections-based method is plotted for different values of the trimming distance, along with the best possible ideal instance of the method, shown with an asterisk in the graph. The ANN curve is also plotted for several values of ϵ.

measure of precision that has been used is the *average rank number*, computed as follows: for each query point a sequence of prototypes, sorted from smaller to larger distances to the point, has been built (using an exhaustive search); then, the sequence number in that sorted list of the point selected by the approximate algorithm as the nearest neighbour is the *rank number*. The *average rank number* for the 10.000 test points has been computed for each parameter setting of the search methods. As it was expected, the results (see Figures 2 & 3) show again better performance of the *kd*-tree+ANN method over the Projections approximation.

Again, the performance of the Nene & Nayar method with ideal parameters is only moderately better than the approximate *kd*-tree search, leaving little space for possible developments in the estimation of ϵ_d that improve the baseline implementation. However, adaptive methods to set the values of ϵ_d as a function of the region of the space where the query point is located can be used to try to get nearer the ideal performance. In the normally ditributed data-set, however, the room for improvement is a little larger than in the uniform distribution.

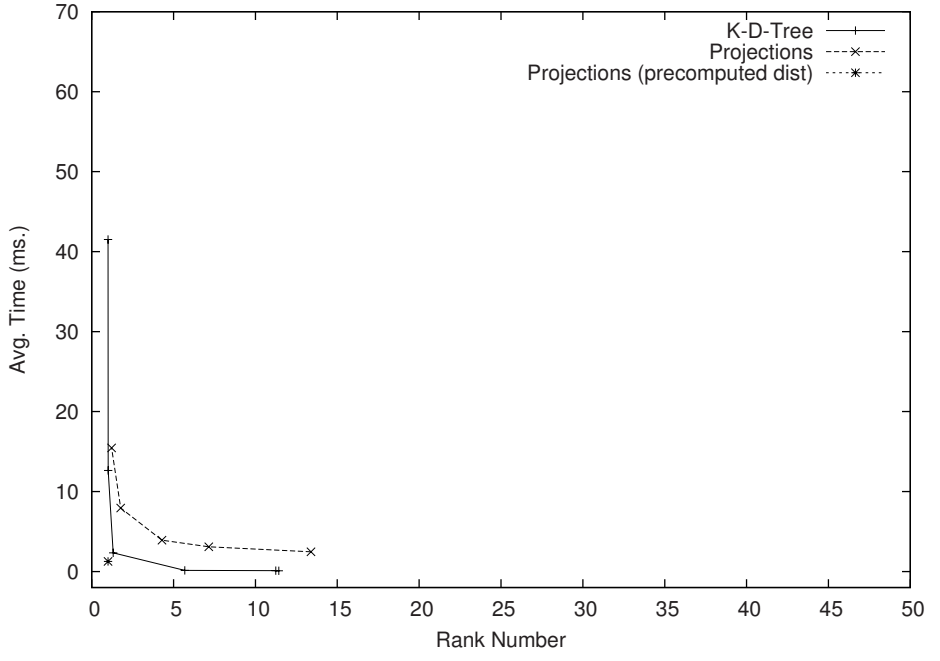

Fig. 2. *kd*-tree+ANN vs. Projections with uniformly distributed synthetic data; a *rank number* = 1.0 means the same result as an exact search. The curves are plotted for different values of ϵ. The ideal parameter setting for the projections method is shown as an asterisk.

5 Conclusions

In this work, two different methods aimed at solving the problem of efficient approximate search of Nearest Neighbours have been compared to find out what is the best option when dealing with high-dimensional and large-sized databases.

As it has been shown in the experiments on classification with real data, *kd*-tree+ANN average times for approximate search are not significantly improved by those obtained by the projection based approximation, even in the best possible (ideal) conditions. For instance, the lower bound found out (with precomputed values of the best possible ϵ_d) for the average time search in the Projection based approximation (0.95 ms.) is not better than the performance achieved with approximate search on the *kd*-tree+ANN, where at the expense of a little increment on the error rate (from 12.13% to 12.32%) an average search time of 0.45 ms. can be obtained.

The results with synthetic data, where the search precision has been directly measured instead of the classification error, confirm that the *kd*-tree+ANN method outperforms the Nene & Nayar technique with fixed values of ϵ_d, and that the ideal adaptive setting of these parameters leaves little space for potentially useful improvements in their estimation, although it cannot be ruled out

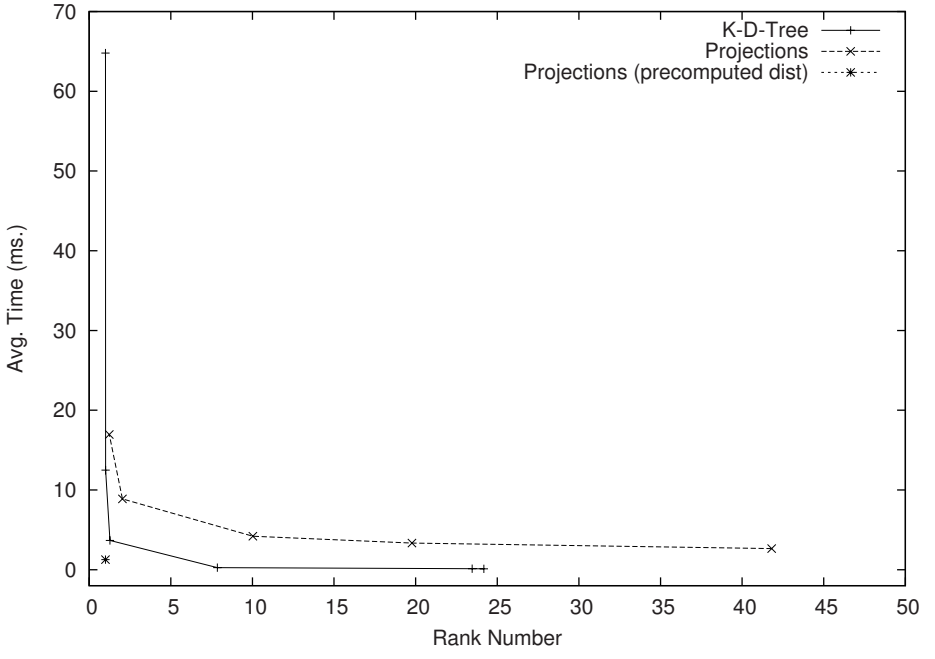

Fig. 3. *kd*-tree+ANN vs. Projections with normaly distributed synthetic data; a *rank number* = 1.0 means the same result as an exact search. The curves are plotted for different values of ϵ. The ideal parameter setting for the projections method is shown as an asterisk.

that new methods to find good values of ϵ_d, adapted to the query region could rival the performance of Approximate Nearest Neighbour search in *kd*-trees.

References

1. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.
2. J. Cano and J.C. Perez-Cortes. Vehicle license plate segmentation in natural images. In *Proceedings of the 1st Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA)*, volume 2652 of *Lecture Notes in Computer Science*, pages 142–149, Puerto de Andratx (Mallorca, Spain), 2003.
3. J. Cano, J.C. Perez-Cortes, J. Arlandis, and R. Llobet. Training set expansion in handwritten character recognition. In *International Workshop on Statistical Pattern Recognition SPR-2002*, volume 2396 of *Lecture Notes in Computer Science*, pages 548–556, Windsor (Ontario, Canada), 2002.
4. J.H. Friedman, F. Baskett, and L.J. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 25(10):1000–1006, 1975.
5. S. A. Nene and S. K. Nayar. A simple algorithm for nearest neighbor search in high dimensions. *IEEETPAMI: IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19, 1997.