The Theory and Practice of Similarity Searches in High Dimensional Data Spaces[*] Extended Abstract

Roger Weber
ETHZ
Zurich, Switzerland
weber@inf.ethz.ch

Pavel Zezula[†]
CNUCE-CNR
Pisa, Italy
zezula@iei.pi.cnr.it

August 27, 1997

1 Introduction

Similarity search in multimedia databases is typically performed on abstractions of multimedia objects, also called the features, rather than on the objects themselves. Though the feature extraction process is application specific, the resulting features are most often considered as points in high-dimensional vector spaces (e.g. the color indexing method of Stricker and Orengo [SO95]). Similarity (or dissimilarity) is then determined in terms of the distance between two feature vectors.

In order to manage similarity search retrieval in large object bases, several storage structures have been designed. However, most of the practical applications reported have observed the *dimensional curse*, i.e. the rapid performance deterioration with the increasing space dimensionality.

In this article, we elaborate on the performance issue of the similarity searches in high dimensional data spaces. Unless explicitly stated, the following assumptions are respected: (1) objects are described by feature vectors in a d-dimensional vector space, (2) the similarity is measured as the Euclidean distance, (3) there is no correlation between data

[*]This research has been funded by the EC ESPRIT Long Term Research program, project no. 9141, HERMES (Foundations of High Performance Multimedia Information Management Systems). The work of Pavel Zezula has also been supported by Grants GACR No. 102/96/0986, Object-oriented data model, and KONTAKT No. PM96 S028, Parallel text bases. The work of Roger Weber has been funded by the Swiss *Bundesamt für Bildung und Wissenschaft* (BBW, grant no. 93.0135).

[†]On leave from the CVIS, Technical University, Údolni 19, Brno, Czech Republic, E-mail: zezula@cis.vutbr.cz

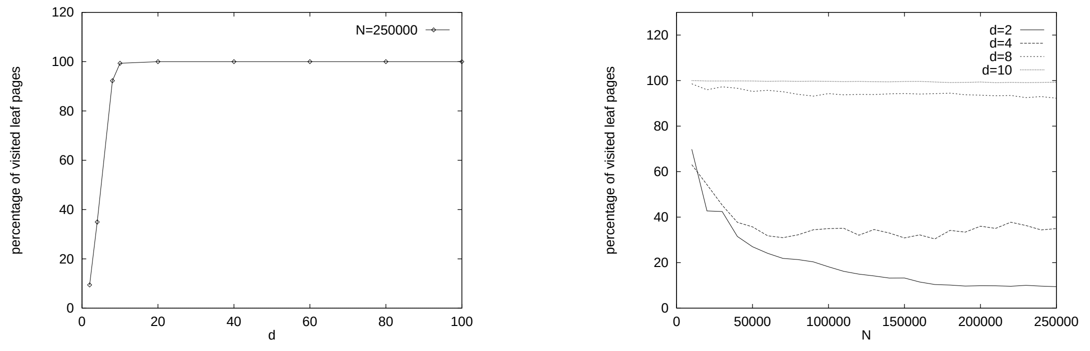


Figure 1: Nearest Neighbor search in R*-Trees

on different levels, and (4) the feature vectors are uniformly distributed in the data space $D = [0,1]^d$.

Given these assumptions, we start with some experiments with R-Tree-like methods [Gut84, BKSS90, BAK96] to confirm the fact, that they completely fail, if the dimensionality goes beyond a small number, say 16.

To better understand why these methods fail, we investigate the nearest neighbor search from a theoretical point of view. Then, recent proposals, based on different partitioning principles, are considered and their performance characteristics investigated with respect to the feature vector dimensionality.

2 The dimensional curse for the R-Tree structures

This section presents some experiments with the R*-Tree [BKSS90], followed by theoretical studies on the nearest neighbor search performance, in general.

2.1 Experiments with R*-Tree

The R*-Tree [BKSS90] is an improved access method of the R-Tree [Gut84]. Both methods partition the data space recursively and store information about the partitions in the nodes. The partitions are described by the minimal bounding box, which covers all objects within the partition. The R-Tree typically consists of a small number of levels (< 10) and a high fill grade of the leaf nodes ($> 70\%$). A provable optimal nearest neighbor search algorithm has been defined by Hjaltson and Samet [HS95] and proved by Berchtold et al. [BBAK97]. It visits the partitions according to their minimal distance to the query point. As soon as a vector has been found, that lies nearer to the query point than all remaining partitions, the nearest neighbor has been found. Generally, the algorithm stops after having found k points—the k nearest vectors to the query. To measure the quality of an access method, one typically counts the number of visited pages. Given the tree structure of the R*-Tree, we assume, that all but the leaf nodes are cached in memory.

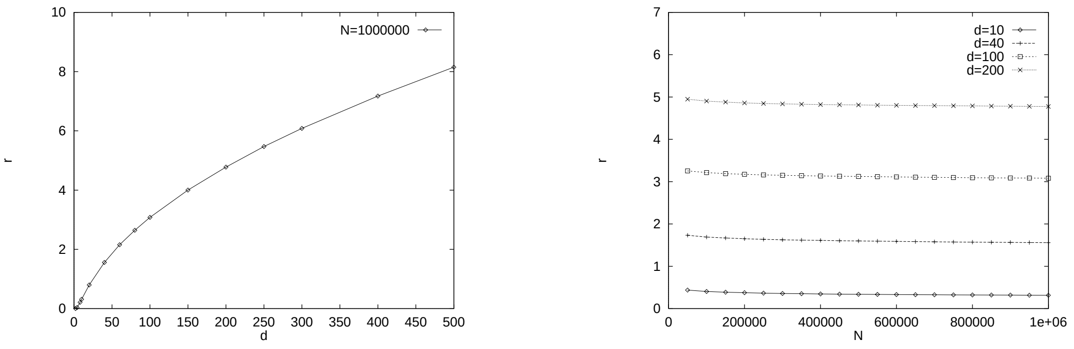


Figure 2: Expected nearest neighbor distance

Therefore, we only measure the number of visited leaf nodes.

Figure 1 shows the number of visited leaf nodes of an R*-Tree for an uniformly distributed data set within the vector space $D = [0,1]^d$, whereas d is the dimension of the space and N is the number of points in the database. The page size for all trees was 8K. One quickly observes, that nearest neighbor search in R*-Tree is hopeless as soon as the dimensionality goes above 10, because all leaf nodes must be visited. Consequently, a simple scan through the vectors would perform better than a search with the R*-Tree.

2.2 Theoretical Studies of the Nearest Neighbor Problem

Similar to [BBAK97], we first computed the expected nearest neighbor distance in high-dimensional vector spaces (see Figure 2). As expected, the distance grows as dimensionality gets larger and shrinks if more points are used.

Then, we computed the number of leaf pages intersecting the query sphere, that is the sphere around the query point with the radius equal to the nearest neighbor distance (see Figure 2). For this purpose, we have to determine the Minkowski sum of the query sphere and the bounding box of the leaf pages. Instead of evaluating the closed formula given in [BBAK97], we focused on the size of the bounding boxes at the leaf nodes. We found out, that the concept of Minkowski sum transforms the leaf pages in enlarged objects, which cover the entire data space. In other words, each of the leaf nodes intersects the query sphere and therefore has to be visited during a nearest neighbor search [1]. To verify this statement, we examined the leaf nodes of R*-Trees and computed the maximal distance of their bounding boxes to any point in the data space. The results together with the expected nearest neighbor distance are shown in Figure 3. The fact, that the maximum distance to any point in the data space is smaller than the expected nearest neighbor distance proves, that the enlarged leaf page incorporates the entire data space. Consequently, all leaf nodes of the R*-Tree must be visited during a nearest neighbor search and the search degrades to a linear problem. Further experiments have shown, that this holds true for every tree-like structure, which uses hyper cubes as bounding

[1]The prove for this and the following statements is subject of a future paper

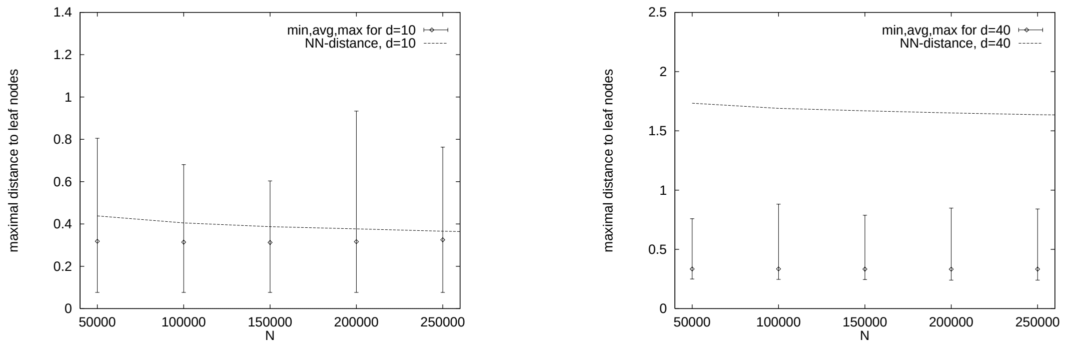


Figure 3: Nearest neighbor distance compared to the maximal distance to leaf nodes; on the left for $d = 10$ and on the right for $d = 40$

boxes.

In literature, it is commonly accepted that the X-Tree [BKSS90] is the most efficient tree structure for nearest neighbor search in high-dimensional vector spaces. Figure 1 and several experiments [BKSS90, BBAK97, BBB$^+$97] show, that the X-tree is better than the R*-Tree, but can not get rid of the dimensional curse too. As long as the dimensionality is low (less than 16) the X-tree can efficiently prune the search space for nearest neighbor queries, but for vectors of higher dimensionality, the complexity of nearest neighbor search becomes $O(n)$, because all of the leaves must be visited.

3 New Approaches

As we have just demonstrated, the nearest neighbor problem is, for uniformly distributed high-dimensional vector sets, linear. Furthermore, experiments have shown that a simple scan through the database outperforms more sophisticated tree-like structures. However, it is possible to reduce the cost of the sequential scan by using small approximations of the vectors, which are stored in a separate, much smaller, file. These approximations may be used to underestimate the distance of the object to the query and to filter out candidates. This method works like the signature method and is called *VA-File*.

The second approach, the so called M-tree, enables nearest neighbor search for metric spaces, which includes the previously discussed vector spaces. The principle difference with this method is that the search space is partitioned only according to distances between objects, i.e. no coordinates of the space are used. From the performance point of view, M-tree aims at reducing not only the I/O, but also the CPU costs.

3.1 The VA-File

Similarly to signature methods, the vector approximation file (*VA-File*) proposed here does not partition data as the R-Tree like methods do. In particular, the data space is

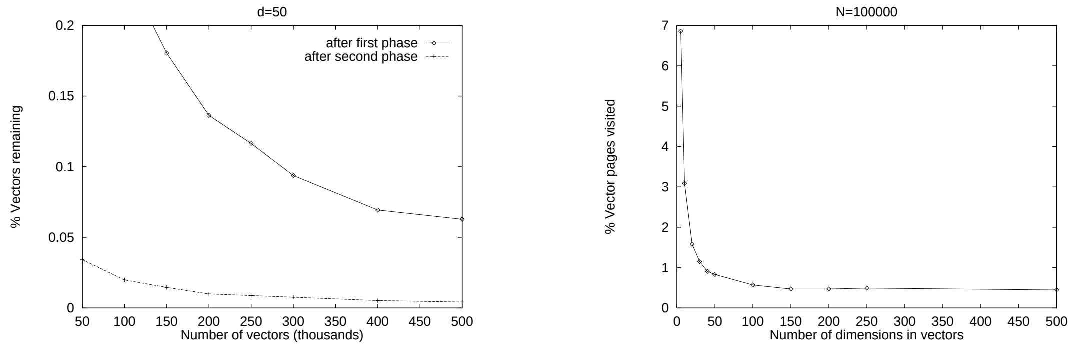


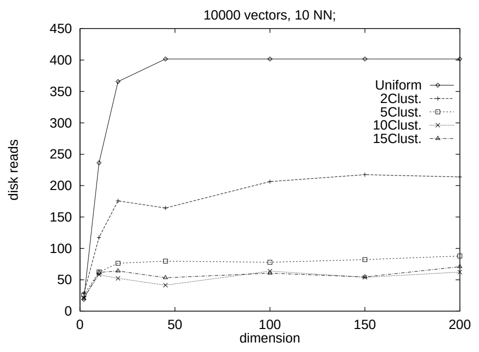
Figure 4: Selectivity as a function of the databases size (left); and page selectivity as a function of d

first partitioned into regions, and then these regions are used to generate bit-encoded approximations for each vector. Contrary to the grid-file [NHS84], no directory over these cells is built, because the number of cells grows exponentially with dimensionality. Rather, all approximations are stored in a sequential file. Queries, vectors in the same data space, are not approximated. To perform nearest neighbor search, the VA-File operates in two phases. In the first phase, it applies a filter. Based on the approximation, it computes for each approximation the minimal and maximal bound on the distance between the query point and the region represented by the approximation. Given that one is looking for the k nearest vectors to the query point, one can select an approximation as a candidate for the second phase if its minimal bound is smaller than the maximal bound of the k-th nearest approximation. In the second phase, the candidates are visited in increasing order of their minimal bound to determine the final answer set. This phase ends when a minimal bound is found which exceeds or equals the k-th best distance in the answer set. Our practical experiments have shown, that between 95% and 99% of the vectors are eliminated during the first filtering step, and 99.9% over all (see Figure 4, left side). Thus, only a small number of vectors must be accessed eventually, and the total number of I/O operations is smaller than with sequential scanning all vectors.

Figure 4 (left) shows the effectiveness of the two filtering steps with vectors of 50 dimensions. The right side of Figure 4 proves, that the VA-File does not suffer from the dimensional curse (the number of vectors is allways 100000). It even gets better with growing dimensionality. Furthermore, in wall-clock experiments, the VA-File was up to three times faster than a simple scan. The larger the database was, the better the method performed.

3.2 The M-tree

In [PP97], a paged metric tree, called M-tree, has been designed. In order to organize and partition the search space, this approach only considers relative distances, rather than absolute positions, of objects in a multi-dimensional space. The M-tree is a balanced tree, able to deal with dynamic data files, and as such it does not require periodical reor-



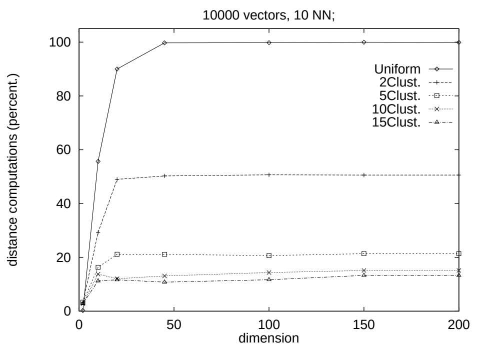


Figure 5: Selectivity as a function of the databases size (left); and page selectivity as a function of d

ganizations. M-tree can also index objects using features compared by distance functions which either do not fit into a vector space or do not use an L_p metric, thus considerably extends the cases for which efficient query processing is possible. What is only required for this approach to work is that the function used to measure the distance, or better to say the dissimilarity, between objects is a *metric*, so that the *triangle inequality* property applies and can be used to prune the search space. Naturally, the M-tree can support both the range and the k-nearest neighbor queries.

As expected, M-tree does suffer from the dimensional curse provided the data is independent and uniformly distributed in n-dimensional space. With such files, the 10 nearest neighbor (NN) queries on 10-dimensional vectors already require more than 50% of both the page reads as well as the distance computations – vectors of 20 dimensions push this percentage up to 90%, and the search is becoming practically linear for $n = 25$. The results are presented in Figure 5.

However, interesting behaviour can be observed for "clustered" feature files – real life files are rarely uniform and independent, thus clusters of objects can be recognized (see Figure 5). With only just five clusters, the number of necessary distance computations has been upper bounded by about 20% of the vectors in the file, and this was true for any file with dimensionalities between 20 and 200 – 200 dimensions was the maximum dimensionality we have tried. Similar behaviour has been observed for the necessary number of page reads.

4 Conclusions

The performance problem of similarity searches in high dimensional data spaces has been elaborated. It has been demonstrated, both by experiments and the theoretical analysis, that the tree oriented search structures used for similarity retrieval significantly suffer from the dimensionality curse, provided the vectors are independent and uniformly distribute. Since, in such situation, the performance complexity seems to be linear, the sequential search on the vector file, or, better, on its approximation, such as the VA-file, seems to be the best solution.

However, as soon as the feature vectors start to form clusters, the performance of tree-based structures can significantly improve. At least such behaviour has been observed for the M-tree.

References

[BAK96] Stefan Berchtold, Daniel A.Keim, and Hans-Peter Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 28–39, 1996.

[BBAK97] Stefan Berchtold, Christian Böhm, Daniel A.Keim, and Hans-Peter Kriegel. A cost model for nearest neighbour search. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 78–86, Tucson, Arizon USA, 1997.

[BBB$^+$97] Stefan Berchtold, Christian Böhm, Bernhard Braunmüller, Daniel A.Keim, and Hans-Peter Kriegel. Fast parallel similarity search in multimedia databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1–12, Tucson, Arizon USA, 1997.

[BKSS90] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In Hector Garcia-Molina and H. V. Jagadish, editors, *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, 23–25 May 1990. *SIGMOD Record* 19(2), June 1990.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. In B. Yormack, editor, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 47–57, Boston, MA, June 1984. acm.

[HS95] G. R. Hjaltason and H. Samet. Ranking in spatial databases. *Lecture Notes in Computer Science*, 951:83–??, 1995.

[NHS84] 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.

[PP97] M.Patella P.Ciaccia and P.Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the International Conference on Very Large Databases (VLDB)*, Athens, Greece, 1997.

[SO95] Markus Stricker and Markus Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases, SPIE*, San Jose, CA, 1995.