

## DDS: An Efficient Dynamic Dimension Selection Algorithm for Nearest Neighbor Search in High Dimensions

Chia-Chen Kuo and Ming-Syan Chen  
Electrical Engineering Department  
National Taiwan University  
Taipei, Taiwan, ROC

### Abstract

*The Nearest Neighbor Search problem is defined as follows: given a set  $P$  of  $n$  points, answer queries for finding the closest point in  $P$  to the query point. This problem arises in a large variety of multimedia applications, particularly in the context of similarity search. In the past few years, there has been increasing interest in performing similarity search over high dimensional data especially for multimedia applications. Unfortunately, most well-known techniques for solving this problem suffer from the "curse of dimensionality" that means the performance of system scales poorly with increased dimensionality of underlying data. The refined algorithms typically achieve a query time that is logarithmic in the quantity of points and exponential in the number of dimensions. However, once the number of dimension exceeds 15, searching in  $k$ - $d$  trees or related structures involves the examination of a large fraction of the search space, thereby performing no better than exhaustive search. In view of this, we propose an efficient dynamic dimension selection algorithm to improve the performance of the nearest neighbor search especially in high dimensions.*

### 1. Introduction

The problem of nearest neighbor search can be defined as follows: given a database of  $n$  points in some metric feature space, answer queries that required for finding the point in the database closest to the query point. Recently, this issue has drawn much attention since the similarity search is broadly utilized in various aspects. Examples of such areas include multimedia applications [1], information retrieval [2], data mining [3], and machine learning [4].

Typically, all data objects are mapped to a multi-dimensional space with lots of features, and similarity search is simplified to the nearest neighbor search in the feature space. The resulting feature space requires very high dimensions especially for multimedia applications [5]. For instance, images are mapped to a set of features containing color, shape, texture and related information in image retrieval systems such as IBM's QBIC [6]. The

dimensionality of involved features is normally large enough to reach hundreds of dimensions. In the meanwhile, the dimensionality is expected to be still high as an increasing number of techniques for feature extractions are being developed. Similar examples were also shown in the domain of pattern recognition and video retrieval. It is undoubtedly that the dimensionality of search spaces will increase as systems and data become more huge and complicated in the future. Thus the demand for an efficient solution for the nearest neighbor search problem in high dimensions becomes very imperative.

The nearest neighbor search problem has been solved efficiently in low dimensions; however, there has been little progress made for high dimensionality. Most approaches either build index structures that require storage space exponential in the number of dimensions, or require prohibitive query time which is close to that required by a thorough scan of all points. Indeed, once the number of dimensions is larger than the logarithm of quantity of points, the exhaustive search is usually the best choice both in theory and in practice [7]. From this point of view, we propose an efficient approach to improve the performance of the nearest neighbor search in high dimensional databases. Our approach is devoted to improve the efficiency of exhaustive search based on dynamically filtering out redundant dimensions for reading and calculating in high dimensions. For this purpose, we devise a dynamic dimension selection algorithm, referred to as DDS, adapted to different distributions of data in order to increase the flexibility for a variety of multimedia applications. It is worth mentioning that our algorithm is simple and memory efficient. In consequence of this, it is feasible to implement our approach in hardware using inexpensive components. Moreover, the performance evaluation shows high performance gains by DDS compared to conventional methods. This is the very advantage we have by developing algorithm DDS for nearest neighbor search in high dimensions.

This paper is organized as follows. The background and related work are introduced in Section 2. Algorithm DDS is presented in Section 3. In addition, the

performance evaluation of our approach will be addressed in Section 4. Finally, this paper concludes with Section 5.

## 2. Background and Related Work

Most of the similarity measures, particularly for multimedia data, are based on underlying vector spaces. In such kind of spaces, the number of features is usually figured as the number of dimensions. For multimedia applications with vast features, dimensionality is generally very high and this trend appears to be accelerating in the future.

### 2.1. Problem Definition

Let  $P = \{p_1, p_2, \dots, p_n\}$  be a set of  $n$  points in a metric space  $M$  consisting of a  $d$  dimensional vector space  $V$ . Given a set  $P$  of  $n$  points in  $M$ , and a query point  $q \in V$ , a nearest neighbor of  $q$  is a point  $p \in P$ , such that for any other point  $r \in P$ , distance  $d(p, q) \leq d(r, q)$

Nearest neighbor search is the problem of returning nearest neighbor of the query point  $q$ . The performance is normally evaluated by the average response time of query processing as well as the storage space of index structures.

### 2.2. Related Work

Nearest neighbor search algorithms are generally divided into three categories: (a) exhaustive search, (b) hashing search, (c) partitioning search [8][9]. Exhaustive search, as the term implies, calculates the distance of the query point from each point in the search space and finds the point with the minimum distance. The complexity of this algorithm is clearly  $O(nd)$  and inefficient for low dimensionality. Hashing search is usually the fastest search technique and runs in constant time. However, the required space to store the hashing table increases exponentially with dimensionality and rapidly becomes impractical for large databases in multimedia applications. The most widely devised algorithm for searching in multiple dimensions is the partitioning search technique based on dimensional binary search trees. A typical representation of partition search algorithm is the  $k$ - $d$  tree. For specific dimension and under certain assumptions about the underlying data, the  $k$ - $d$  tree requires  $O(n \log n)$  operations to construct and  $O(\log n)$  operations to search. The  $k$ - $d$  tree or related index structures is very efficient to employ, particularly in low dimensions.

### 2.3. Curse of Dimensionality

Partitioning methods generally work well in low dimensionality. However, the performance often degrades drastically with increased dimensions. This phenomenon has been generally called the "curse of dimensionality". As dimensionality increases, the Central Limit Theorem proves that the distance among points is approaching a

normal distribution [10]. Figure 1 describes the mean distance increases with dimensions while the standard deviation remains nearly unchanged under uniform distributed vector spaces.

This effect causes serious impacts on partitioning methods because all points begin to appear almost equidistant from each other. It is difficult to effectively prune the search space by partitioning techniques. Hence an efficient approach is required to support the query in high dimensionality.

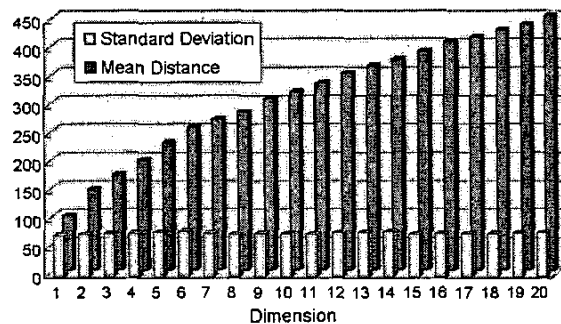


Figure 1: mean distance and standard deviation

## 3. Dynamic Dimension Selection

As mentioned in previous sections, the exhaustive search is usually the best choice both in theory and in practice for the number of dimensions larger than the logarithm of quantity of points. Our approach is devoted to improve the efficiency of exhaustive search based on dynamically filtering out redundant dimensions for reading and calculating in high dimensions.

### 3.1. Filtering out Redundant Dimensions

Most existing methods for nearest neighbor search concentrated on the storage size of index structure or the number of disk access that corresponds to the number of points must to be calculated. Compared with the number of disk access, however, the data size of calculated points becomes more and more important as dimensionality increases. Since the elapsed time of reading and calculating a single point increases with the number of dimensions, it is not efficient to read and compute all dimensions for each point. As validated by our simulation results, it is possible to save considerable time when filtering out redundant dimensions for reading and calculating. Note that each time we read only one dimension of a single point and calculate the distance from the query point cumulatively. Once the cumulate distance exceeds the current minimum distance, it is unnecessary to read other remaining dimensions for computations because this point is impossible to be the closest point. For example, if this approach stops by 30

dimensions in a 64-dimension database, it saves the time of reading and calculating the other 34 dimensions.

Figure 2 demonstrates the average time saved by filtering out redundant dimensions for reading and calculating in uniform distributed vector spaces. Our simulation results show the percentage of time saved for four different quantities of points ( $10^2$ ,  $10^3$ ,  $10^4$ ,  $10^5$ ) and five different dimensions (1, 4, 16, 64, 256). For instance, in a  $10^5$  points 64-dimension uniform distributed database, we can averagely save 50% of time by filtering out redundant dimensions.

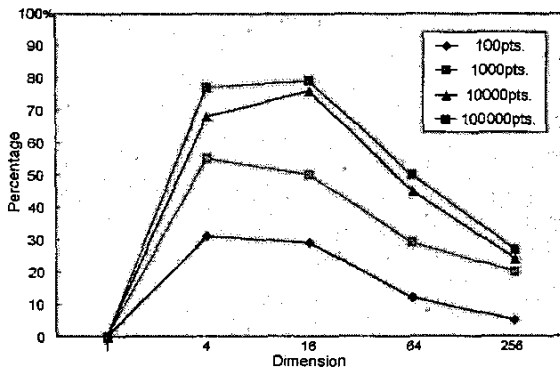


Figure 2: time saved by filtering out redundant dimension

### 3.2. Outline of DDS

However, the response time of disk I/O depends on not only the data size but also the number of disk access. It is not efficient to access disk frequently because the additional overhead of reading degrades the overall performance. To solve this problem, we propose algorithm DDS to adaptively predict the number of dimensions required to be read and calculated. Algorithm DDS is devised to achieve a balance between the reading data size and the number of disk access.

As validated by our simulation of uniform distributed vector spaces illustrated in Figure 3, the number of dimensions required for reading and calculating in effect gives a good approximation to the normal distribution. Hence, we set a prediction threshold in order to control the predicted number of dimensions adaptively based on the rate of successful prediction. It is evident that the prediction threshold determines the tradeoff between the reading data size and the number of disk access. Algorithm DDS is outlined as follows:

1. Enter a query point for nearest neighbor search. Decide the *prediction threshold* as the expected rate of successful predictions,  $0 < \text{prediction threshold} \leq 1$ .
2. Initialize (a) *predicted dimension* = maximum dimension, (b) *minimum distance* =  $\infty$ , (c) *failure prediction* = *successful prediction* = 0.
3. Read one point. The number of dimensions required for reading is determined by *predicted dimension*.

Calculate the distance from the query point by the data of read dimensions.

4. If distance  $\geq$  *minimum distance*, increase *successful prediction* by one. It implies that we save the time by filtering out redundant dimensions.
5. If distance  $<$  *minimum distance*, increase *failure prediction* by one and read all other remaining dimensions. Calculate the distance from the query point by all dimensions. If distance  $<$  *minimum distance*, update *minimum distance* with new distance.
6. Compute *successful ratio* = *successful prediction* / (*successful prediction* + *failure prediction*).
7. If *successful ratio*  $<$  *prediction threshold*, increase *predicted dimension* by one.
8. If *successful ratio*  $>$  *prediction threshold*, decrease *predicted dimension* by one.
9. Back to *step 3* until all points are calculated.

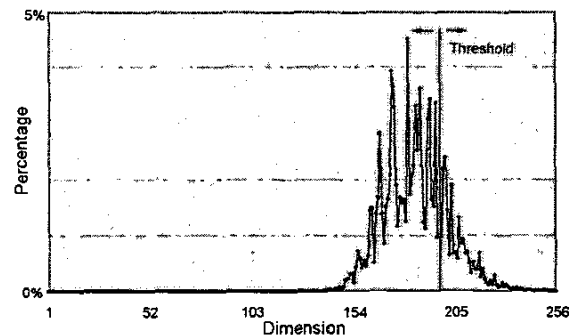


Figure 3: dimension distribution of 256-dimension data

## 4. Performance Evaluation

In this section, we will present experimental results for benchmarks performed on statistically structured data and explore the effects of the *prediction threshold* parameter. Our test platform is equipped with Pentium 4 2GHz and 512MB RAM. Without loss of generality, ten uniform distributed sets containing 100,000 points are used. For each algorithm, the execution time is estimated by averaging the total time required to perform a nearest neighbor search on each of the 100,000 points in this test sets. It is noted that a major advantage of algorithm DDS is its simplicity and efficient use of memory. The main computations performed by DDS are simple squares and integer comparisons to calculate distance. Consequently, it is feasible to implement DDS in hardware using inexpensive components.

### 4.1. Benchmarks

Algorithm DDS is compared with typical k-d tree and exhaustive search algorithms. Figure 4 exhibits the average execution time per search under different dimensions. It is obvious that algorithm DDS with *prediction threshold* 0.7 is faster than the k-d tree

algorithm for dimensions exceeding 8. Although the execution time of algorithm DDS with *prediction threshold* 0.5 rises, it is still faster than the k-d tree algorithm for dimensions exceeding 12. Also notice that the k-d tree algorithm actually runs slower than exhaustive search for dimensions exceeding 16. This phenomenon can be explained as follows. In high dimensions, the space is so sparsely occupied that the radius of query hypersphere grows very large. Since the hypersphere intersects almost all index nodes, a large quantity of points must be calculated. Also, the additional overhead of traversing the tree structure makes it very inefficient to search in high dimensions.

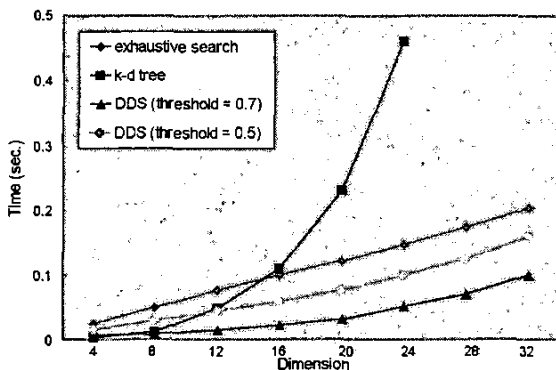


Figure 4: average execution time per search

#### 4.2. Effects of Prediction Threshold

From the previous evaluation, we have observed the impact of the *prediction threshold* parameter. This parameter determines the tradeoff between the reading data size and the number of disk access. Figure 5 shows the effects of different *prediction threshold* to the average execution time under 20-dimension uniform distributed test sets. As shown in our experiments, we have a good result when we set the prediction threshold as 0.7 under uniform distribution. Clearly, we can pursue better *prediction threshold* for other distributions or multimedia database systems by verifying the average execution time in advance. Furthermore, we are able to develop an algorithm to adjust the prediction threshold adaptively according to the minimum execution time.

#### 5. Conclusion

In this paper, we proposed algorithm DDS for nearest neighbor search in high dimensions that is usually required for multimedia applications. DDS is based on dynamically filtering out redundant dimensions to improve the execution efficiency. We conducted the performance evaluation of algorithm DDS and discussed

the effect of resident parameters. The experimental results validate high performance gains by DDS compared to conventional methods especially in high dimensions.

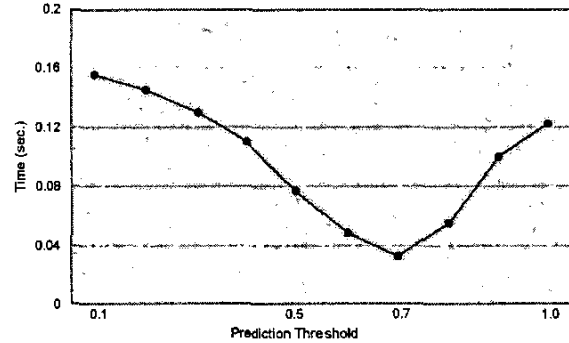


Figure 5: average execution time per search

#### 6. References

- [1] P. Wu and B. Manjunath, "Adaptive nearest neighbor search for relevance feedback in large image datasets," *Proceedings of ACM Multimedia*, pp. 87-98, 2001.
- [2] P. Raghavan, "Information retrieval algorithms: a survey," *ACM-SIAM Symposium on Discrete Algorithms*, pp. 11-18, 1997.
- [3] C. Bohm and F. Krebs, "High performance data mining using the nearest neighbor join," *IEEE International Conference on Data Mining*, pp. 43-50, 2002.
- [4] S. Cost and S. Salzberg, "A weighted nearest neighbor algorithm for learning with symbolic features," *Machine Learning*, Vol. 10, pp. 57-67, 1993.
- [5] S. Brin, "Near neighbor search in large metric spaces," *Proceedings of International Conference on Very Large Data Bases*, pp. 574-584, 1995.
- [6] M. Flickner, H. Sawhney, W. Niblack, et. al, "Query by image content: the QBIC system," *IEEE Computer*, Vol. 28, No. 9, pp. 23-32, 1995.
- [7] R. Weber, H. Schek, and S. Blott, "A quantitative analysis and performance study for similarity search methods in high-dimensional spaces," *Proceedings of International Conference on Very Large Data Bases*, pp. 194-205, 1998.
- [8] S. Nene and S. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No.9, pp. 989-1003, 1997.
- [9] P. Yianilos, "Data structures and algorithms for nearest neighbor in general metric spaces," *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, pp. 311-321, 1993.
- [10] S. Blott and R. Weber, "A simple vector-approximation file for similarity search in high-dimensional vector spaces," *Technical Report, Institute for Information Systems, Zurich, Switzerland*, 1997.