

# A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces

Roger Weber

Institute of Information Systems  
ETH Zentrum, 8092 Zurich  
weber@inf.ethz.ch

Hans-J. Schek

Institute of Information Systems  
ETH Zentrum, 8092 Zurich  
schek@inf.ethz.ch

Stephen Blott

Bell Laboratories (Lucent Technologies)  
700 Mountain Ave, Murray Hill  
blott@research.bell-labs.com

## Abstract

For similarity search in high-dimensional vector spaces (or ‘HDVSs’), researchers have proposed a number of new methods (or adaptations of existing methods) based, in the main, on data-space partitioning. However, the performance of these methods generally degrades as dimensionality increases. Although this phenomenon—known as the ‘dimensional curse’ is well known, little or no quantitative analysis of the phenomenon is available. In this paper, we provide a detailed analysis of partitioning and clustering techniques for similarity search in HDVSs. We show formally that these methods exhibit linear complexity at high dimensionality, and that existing methods are outperformed on average by a simple sequential scan if the number of dimensions exceeds around 10. Consequently, we come up with an alternative organization based on *approximations* to make the unavoidable sequential scan as fast as possible. We describe a simple *vector approximation* scheme, called *VA-file*, and report on an experimental evaluation of this and of two tree-based index methods (an R\*-tree and an X-tree).

## 1 Introduction

An important paradigm of systems for multimedia, decision support and data mining is the need for simi-

---

*Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.*

Proceedings of the 24th VLDB Conference  
New York, USA, 1998

ilarity search, i.e. the need to find a small set of objects which are similar or close to a given query object. Mostly, similarity is not measured on the objects directly, but rather on abstractions of objects termed *features*. In many cases, features are points in some high-dimensional vector space (or ‘HDVS’), as such they are termed *feature vectors*. The number of dimensions in such feature vectors varies between moderate, from 4–8 in [19] or 45 in [32], and large, such as 315 in a recently-proposed color indexing method [13], or over 900 in some astronomical indexes [12]. The similarity of two objects is then assumed to be proportional to the similarity of their feature vectors, which is measured as the distance between feature vectors. As such, similarity search is implemented as a nearest neighbor search within the feature space.

The conventional approach to supporting similarity searches in HDVSs is to use a multidimensional index structure. Space-partitioning methods like grid-file [27], K-D-B-tree [28] or quadtree [18] divide the data space along predefined or predetermined lines regardless of data clusters. Data-partitioning index trees such as R-tree [21], R<sup>+</sup>-tree [30], R\*-tree [2], X-tree [7], SR-tree [24], M-tree [10], TV-tree [25] or hB-tree [26] divide the data space according to the distribution of data objects inserted or loaded into the tree. Bottom-up methods, also called clustering methods, aim at identifying clusters embedded in data in order to reduce the search to clusters that potentially contain the nearest neighbor of the query. Several surveys provide background and analysis of these methods [1, 3, 15, 29]. Although these access methods generally work well for low-dimensional spaces, their performance is known to degrade as the number of dimensions increases—a phenomenon which has been termed the *dimensional curse*. This phenomenon has been reported for the R\*-tree [7], the X-tree [4] and the SR-tree [24], among others.

In this paper, we study the performance of both space- and data-partitioning methods at high dimensionality from a theoretical and practical point of view.

Under the assumptions of uniformity and independence, our contribution is:

- We establish lower bounds on the average performance of existing partitioning and clustering techniques. We demonstrate that these methods are outperformed by a sequential scan whenever the dimensionality is above 10.
- By establishing a general model for clustering and partitioning, we formally show that there is no organization of HDVS based on clustering or partitioning which does not degenerate to a sequential scan if dimensionality exceeds a certain threshold.
- We present performance results which support our analysis, and demonstrate that the performance of a simple, approximation-based scheme called *vector approximation file* (or ‘VA-file’) offers the best performance in practice whenever the number of dimensions is larger than around 6. The VA-File is the only method which we have studied for which performance can even improve as dimensionality increases.

The remainder of this paper is structured as follows. Section 2 introduces our notation, and discusses some properties of HDVSs which make them problematic in practice. Section 3 provides an analysis of nearest neighbor similarity search problems at high dimensionality, and demonstrates that clustering and partitioning approaches degenerate to a scan through all blocks as dimensionality increases. Section 4 sketches the VA-File, and provides a performance evaluation of four methods. Section 5 concludes.

## Related Work

There is a considerable amount of existing work on cost-model-based analysis in the area of HDVSs. Early work in this area did not address the specific difficulties of high dimensionality [11, 20, 31]. More recently, Berchtold et al. [5] have addressed the issue of high dimensionality, and, in particular, the boundary effects which impact the performance of high-dimensional access methods. Our use of Minkowski Sums, much of our notation, and the structure of our analysis for rectangular *minimum bounding regions* (or ‘MBRs’) follow their analysis. However, Berchtold et al. use their cost model to predict the performance of particular index methods, whereas we use a similar analysis here to compare the performance of broad classes of index methods with simpler scan-based methods. Moreover, our analysis extends theirs by considering not just rectangular MBRs, but also spherical MBRs, and a general class of clustering and partitioning methods.

There exists a considerable number of reduction methods such as SVD, eigenvalue decomposition, wavelets, or Karhunen-Loève transformation which

can be used to decrease the effective dimensionality of a data set [1]. Faloutsos and Kamel [17] have shown that *fractal dimensionality* is a useful measure of the inherent dimensionality of a data set. We will further discuss this below.

The ‘indexability’ results of Hellerstein et al. [22] are based on data sets that can be seen as regular meshes of extension  $n$  in each dimension. For range queries, these authors presented a minimum bound on the ‘access overhead’ of  $B^{1-\frac{1}{d}}$ , which tends toward  $B$  as dimensionality  $d$  increases ( $B$  denotes the size of a block). This ‘access overhead’ measures how many more blocks actually contain points in the answer set, compared to the optimal number of blocks necessary to contain the answer set. Their result indicates that, for any index method at high dimensionality, there always exists a worst case in which  $|Q|$  different blocks contain the  $|Q|$  elements of the answer set (where  $|Q|$  is the size of the answer set for some range-query  $Q$ ). While Hellerstein et al. have established worst-case results, ours is an average-case analysis. Moreover, our results indicate that, even in the average case, nearest-neighbor searches ultimately access *all* data blocks as dimensionality increases.

The VA-File is based on the idea of object approximation, as it has been used in many different areas of computer science. Examples are the Multi-Step approach of Brinkhoff et al. [8, 9], which approximates object shapes by their minimum bounding box, the signature-file for partial match queries in text documents [14, 16], and multi-key hashing schemes [33]. Our approach of using geometrical approximation has more in common with compressing and quantization schemes where the objective is to reduce the amount of data without losing too much information.

## 2 Basic Definitions and Simple Observations

This section describes the assumptions, and discusses their relevance to practical similarity-search problems. We also introduce our notation, and describe some basic and well-known observations concerning similarity search problems in HDVSs. The goal of this section is to illustrate why similarity search at high dimensionality is more difficult than it is at low dimensionality.

### 2.1 Basic Assumptions And Notation

For simplicity, we focus here on the unit hyper-cube and the widely-used  $L_2$  metric. However, the structure of our analysis might equally be repeated for other data spaces (e.g. unit hyper sphere) and other metrics (e.g.  $L_1$  or  $L_\infty$ ). Table 1 summarizes our notation.

#### Assumption 1 (Data and Metric)

A  $d$ -dimensional data set  $\mathcal{D}$  lies within the unit hyper-cube  $\Omega = [0, 1]^d$ , and we use the  $L_2$  metric (Euclidean metric) to determine distances.

$d$	number of dimensions
$N$	number of data points
$\Omega = [0, 1]^d$	data space
$\mathcal{D} \subseteq \Omega$	data set
$P[\bullet]$	probability function
$E[\bullet]$	expectation value
$sp^d(C, r)$	$d$ -dim sphere around $C$ with radius $r$
$k$	number of NNs to return
$nn(Q)$	NN to query point $Q$
$nn^{dist}(Q)$	NN-distance of query point $Q$
$nn^{sp}(Q)$	NN-sphere, $sp^d(Q, nn^{dist}(Q))$
$E[nn^{dist}]$	expected NN-distance

Table 1: Notational summary

### Assumption 2 (Uniformity and Independence)

Data and query points are uniformly distributed within the data space, and dimensions are independent.

To eliminate correlations in data sets, we assume that one of a number of reduction methods—such as SVD, eigenvalue decomposition, wavelets or Karhunen-Loève—has been applied [1]. Faloutsos and Kamel have shown that the so-called fractal dimensionality can be a useful measure for predicting the performance of data-partitioning access methods [17]. Therefore we conjecture that, for  $d$ -dimensional data sets, the results obtained here under the uniformity and independence assumptions generally apply also to arbitrary higher-dimensional data sets of fractal dimension  $d$ . This conjecture appears reasonable, and is supported by the experimental results of Faloutsos and Kamel for the effect of fractal dimensionality on the performance of R-trees [17].

The nearest neighbor to a query point  $Q$  in a  $d$ -dimensional space is defined as follows:

#### Definition 2.1 (NN, NN-distance, NN-sphere)

Let  $\mathcal{D}$  be a set of  $d$ -dimensional points. Then the nearest neighbor (NN) to the query point  $Q$  is the data point  $nn(Q) \in \mathcal{D}$ , which lies closest to  $Q$  in  $\mathcal{D}$ :

$$nn(Q) = \{P \in \mathcal{D} \mid \forall P' \in \mathcal{D} : \|P - Q\|_2 \leq \|P' - Q\|_2\}$$

where  $\|\bullet - \bullet\|_2$  denotes Euclidean distance. The nearest neighbor distance  $nn^{dist}(Q)$  is the distance between  $Q$  and its nearest neighbor  $nn(Q)$ , i.e.  $nn^{dist}(Q) = \|nn(Q) - Q\|_2$ , and the NN-sphere  $nn^{sp}(Q)$  is the sphere with center  $Q$  and radius  $nn^{dist}(Q)$ .  $\square$

Analogously, one can define the  $k$ -nearest neighbors to a given query point  $Q$ . Then  $nn^{dist,k}(Q)$  is the distance of the  $k$ -th nearest neighbor, and  $nn^{sp,k}(Q)$  is the corresponding NN-sphere.

## 2.2 Probability and Volume Computations

Let  $Q$  be a query point, and let  $sp^d(Q, r)$  be the hyper-sphere around  $Q$  with radius  $r$ . Then, under the uni-

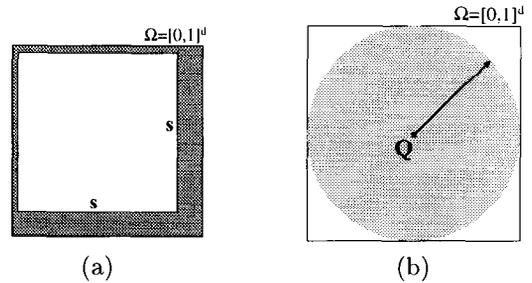


Figure 1: (a) Data space is sparsely populated; (b) Largest range query entirely within the data space.

formity and independence assumptions, for any point  $P$ , the probability that  $sp^d(Q, r)$  contains  $P$  is equal to the volume of that part of  $sp^d(Q, r)$  which lies inside the data space. This volume can be obtained by integrating a piecewise defined function over  $\Omega$ .

$$Vol(sp^d(Q, r) \cap \Omega) = \int_{P \in \Omega} \begin{cases} 1 & \text{if } \|Q - P\|_2 \leq r \\ 0 & \text{otherwise} \end{cases} dP$$

As dimensionality increases, this integral becomes difficult to evaluate. Fortunately, good approximations for such integrals can be obtained by the Monte-Carlo method, i.e. generating random experiments (points) within the space of the integral, summing the values of the function for this set of points, and dividing the sum by the total number of experiments.

## 2.3 The Difficulties of High Dimensionality

The following basic observations shed some light on the difficulties of dealing with high dimensionality.

#### Observation 1 (Number of partitions)

The most simple partitioning scheme splits the data space in each dimension into two halves. With  $d$  dimensions, there are  $2^d$  partitions. With  $d \leq 10$  and  $N$  on the order of  $10^6$ , such a partitioning makes sense. However, if  $d$  is larger, say  $d = 100$ , there are around  $10^{30}$  partitions for only  $10^6$  points—the overwhelming majority of the partitions are empty.

#### Observation 2 (Data space is sparsely populated)

Consider a hyper-cube range query with length  $l$  in all dimensions as depicted in Figure 1(a). The probability that a point lies within that range query is given by:

$$P^d[s] = s^d \quad (1)$$

Figure 2 plots this probability function for some  $l$  as a function of dimensionality. It follows directly from the formula above that even very large hyper-cube range queries are not likely to contain a point. At  $d = 100$ , a

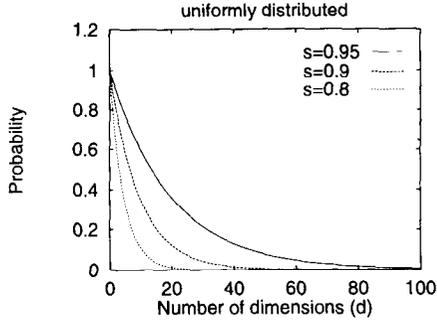


Figure 2: The probability function  $P^d[s]$ .

range query with length 0.95 only selects 0.59% of the data points. Notice that the hyper cube range can be placed anywhere in  $\Omega$ . From this we conclude that we hardly can find data points in  $\Omega$ , and, hence, that the data space is sparsely populated.

**Observation 3** (Spherical range queries)

The largest spherical query that fits entirely within the data space is the query  $sp^d(Q, 0.5)$ , where  $Q$  is the centroid of the data space (see Figure 1(b)). The probability that an arbitrary point  $R$  lies within this sphere is given by the spheres volume:<sup>1</sup>

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{Vol(sp^d(Q, \frac{1}{2}))}{Vol(\Omega)} = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{\Gamma(\frac{d}{2} + 1)} \quad (2)$$

If  $d$  is even, then this probability simplifies to

$$P[R \in sp^d(Q, \frac{1}{2})] = \frac{\sqrt{\pi^d} \cdot (\frac{1}{2})^d}{(\frac{d}{2})!} \quad (3)$$

Table 2 shows this probability for various numbers of dimensions. The relative volume of the sphere shrinks markedly as dimensionality grows, and it increasingly becomes improbable that any point will be found within this sphere at all.

**Observation 4** (Exponentially growing DB size)

Given equation (2), we can determine the size a data set would have to have such that, on average, at least one point falls into the sphere  $sp^d(Q, 0.5)$  (for even  $d$ ):

$$N(d) = \frac{(\frac{d}{2})!}{\sqrt{\pi^d} \cdot (\frac{1}{2})^d} \quad (4)$$

Table 2 enumerates this function for various numbers of dimensions. The number of points needed explodes exponentially, even though the sphere  $sp^d(Q, 0.5)$  is the largest one contained wholly within the data space. At  $d = 20$ , a database must contain more than 40 million points in order to ensure, on average, that at least one point lies within this sphere.

$d$	$P[R \in sp^d(Q, 0.5)]$	$N(d)$
2	0.785	1.273
4	0.308	3.242
10	0.002	401.5
20	$2.461 \cdot 10^{-8}$	40'631'627
40	$3.278 \cdot 10^{-21}$	$3.050 \cdot 10^{20}$
100	$1.868 \cdot 10^{-70}$	$5.353 \cdot 10^{69}$

Table 2: Probability that a point lies within the largest range query inside  $\Omega$ , and the expected database size.

**Observation 5** (Expected NN-distance) Following Berchtold et al. [4], let  $P[Q, r]$  be the probability, that the NN-distance is at most  $r$  (i.e. the probability that  $nn(Q)$  is contained in  $sp^d(Q, r)$ ). This probability distribution function is most easily expressed in terms of its complement, that is, in terms of the probability that all  $N$  points lie outside the hyper-sphere:

$$P[Q, r] = 1 - \left(1 - Vol\left(sp^d(Q, r) \cap \Omega\right)\right)^N \quad (5)$$

The expected NN-distance for a query point  $Q$  can be obtained by integrating over all radii  $r$ :

$$E[Q, nn^{dist}] = \int_0^\infty r \cdot \frac{\partial P[Q, r]}{\partial r} dr \quad (6)$$

Finally, the expected NN-distance  $E[nn^{dist}]$  for any query point in the data space is the average of  $E[Q, nn^{dist}]$  over all possible points  $Q$  in  $\Omega$ :

$$E[nn^{dist}] = \int_{Q \in \Omega} E[Q, nn^{dist}] dQ \quad (7)$$

Based on this formula, we used the Monte-Carlo method to estimate NN-distances. Figure 3 shows this distance as a function of dimensionality, and Figure 4 of the number of data points. Notice, that  $E[nn^{dist}]$  can become much larger than the length of the data space itself. The main conclusions are:

1. The NN-distance grows steadily with  $d$ , and
2. Beyond trivially-small data sets  $\mathcal{D}$ , NN-distances decrease only marginally as the size of  $\mathcal{D}$  increases.

As a consequence of the expected large NN-distance, objects are widely scattered and, as we shall see, the probability of being able to identify a good partitioning of the data space diminishes.

### 3 Analysis of Nearest-Neighbor Search

This section establishes a number of analytical results as to the average performance of nearest-neighbor search in partitioned and clustered organizations of vector spaces. The main objective of this analysis is to provide formulae that allow us to accurately predict the average cost of NN-searches in HDVSSs. Based on these cost formulae, we show formally, under the assumptions of uniformity and independence, that:

<sup>1</sup> $\Gamma(\cdot)$  is defined by:  $\Gamma(x+1) = x \cdot \Gamma(x)$ ,  $\Gamma(1) = 1$ ,  $\Gamma(\frac{1}{2}) = \sqrt{\pi}$

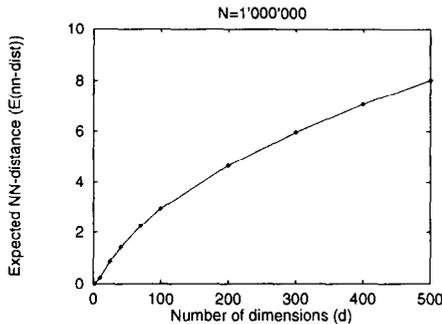


Figure 3:  $E[nn^{dist}]$  as a function of the dimensionality.

- Conventional data- and space-partitioning structures are out-performed by a sequential scan already at dimensionality of around 10 or higher.
- There is no organization of HDVS based on partitioning or clustering which does not degenerate to a sequential scan if dimensionality exceeds a certain threshold.

We first introduce our general cost model, and show that, for practical relevant approaches to space- and data-partitioning, nearest-neighbor search degrades to a (poor) sequential scan. As an important, quantitative consequence, these methods are out-performed by a sequential scan whenever the dimensionality is higher than around 10. Whereas this first result is important from a practical point of view, we second investigate a general class of indexing schemes from a theoretical perspective. Namely, we derive formally that the complexity of any partitioning and clustering scheme converges to  $O(N)$  with increasing dimensionality, and that, ultimately, all objects must be accessed in order to evaluate a nearest-neighbor query.

### 3.1 General Cost Model

For disk-resident databases, we use the number of blocks which must be accessed as a measure of the amount of IO which must be performed, and hence of the ‘cost’ of a query. A nearest-neighbor search algorithm is optimal if the blocks visited during the search are exactly those whose *minimum bounding regions* (MBR) intersect the NN-sphere. Such an algorithm has been proposed by Hjaltson and Samet [23], and shown to be optimal by Berchtold et al. [5]. This algorithm visits blocks in increasing order of their minimal distance to the query point, and stops as soon as a point is encountered which lies closer to the query point than all remaining blocks.

Given this optimal algorithm, let  $M_{visit}$  denote the number of blocks visited. Then  $M_{visit}$  is equal to the number of blocks which intersect the NN-sphere  $nm^{sp}(Q)$  with, on average, the radius  $E[nn^{dist}]$ . To estimate  $M_{visit}$ , we transform the spherical query into a

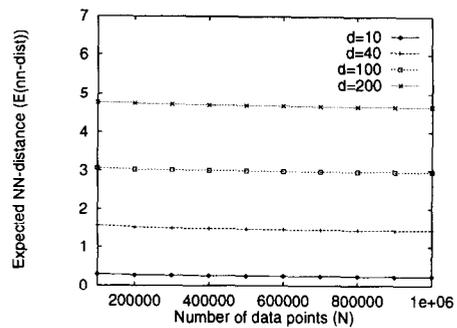


Figure 4:  $E[nn^{dist}]$  as a function of the database size.

point query by the technique of Minkowski sum following [5]. The enlarged object  $MSum(mbr_i, E[nn^{dist}])$  consists of all points that are contained by  $mbr_i$  or have a smaller distance to the surface of  $mbr_i$  than  $E[nn^{dist}]$  (e.g. see in Figure 5, the shaded object on the right hand side). The volume of the part of this region which is within the data space corresponds to the fraction of all possible queries in  $\Omega$  whose NN-spheres intersect the block. Therefore, the probability that the  $i$ -th block must be visited is given by:

$$P_{visit}[i] = Vol \left( MSum \left( mbr_i, E[nn^{dist}] \right) \cap \Omega \right) \quad (8)$$

The expected number of blocks which must be visited is given by the sum of this probability over all blocks. If we assume  $m$  objects per block, we arrive at:

$$M_{visit} = \frac{N}{m} P_{visit}^{avg}, \quad P_{visit}^{avg} = \frac{m}{N} \sum_{i=0}^{N/m} P_{visit}[i] \quad (9)$$

This formula depends upon the geometry of  $mbr_i$ . In the following, we extend this analysis for both the case that MBRs are hyper-rectangles (e.g. R\*-tree and X-tree), and the case that MBRs are hyper-spheres (e.g. TV-tree and M-tree).

In our comparisons, we use a well-tuned sequential scan as a benchmark. Under this approach, data is organized sequentially on disk, and the entire data set is accessed during query processing. In addition to its simplicity, a major advantage of this approach is that a direct sequential scan of the data can expect a significant performance boost from the sequential nature of its IO requests. Although a factor of 10 for this phenomenon is frequently assumed elsewhere (and was observed in our own PC and workstation environments), we assume a more conservative factor of only 5 here. Hence, we consider an index structure to work ‘well’ if, on average, less than 20% of blocks must be visited, and to ‘fail’ if, on average, more than 20% of blocks must be visited.

### 3.2 Space-Partitioning Methods

Space-partitioning methods like gridfiles [27], quad trees [18] and K-D-B-trees [28] divide the data space

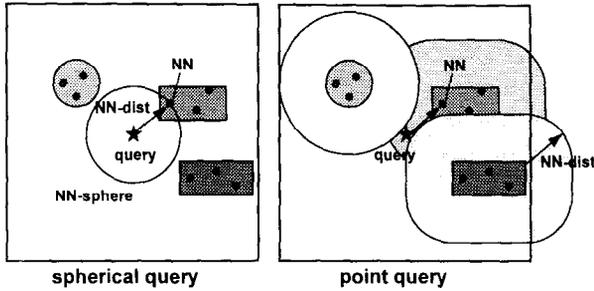


Figure 5: The transformation of a spherical query into a point query (Minkowski sum).

along predefined or predetermined lines regardless of clusters embedded in the data. In this section, we show that either the space consumption of these structures grows exponentially in the number of dimensions, or NN-search results in visiting all partitions.

**Space consumption of index structure:** If each dimension is split once, the total number of partitions is  $2^d$ . Assuming  $B$  bytes for each directory/tree entry of a partition, the space overhead is  $B \cdot 2^d$ , even if several partitions are stored together on a single block.

**Visiting all partitions:** In order to reduce the space overhead, only  $d' \leq d$  dimensions are split such that, on average,  $m$  points are assigned to a partition. Thus, we obtain an upper bound:

$$d' \leq \left\lceil \log_2 \frac{N}{m} \right\rceil \quad (10)$$

Furthermore, each dimension is split at most once, and, since data is distributed uniformly, the split position is always at  $\frac{1}{2}$ .<sup>2</sup> Hence, the MBR of a block has  $d'$  sides with a length of  $\frac{1}{2}$ , and  $d - d'$  sides with a length of 1. For any block, let  $l_{max}$  denote the maximum distance from that block to any point in the data space (see Figure 6 for an example). Then  $l_{max}$  is given by the equation:

$$l_{max} = \frac{1}{2} \sqrt{d'} = \frac{1}{2} \sqrt{\left\lceil \log_2 \frac{N}{m} \right\rceil} \quad (11)$$

Notice that  $l_{max}$  does not depend upon the dimensionality of the data set. Since the expected NN-distance steadily grows with increasing dimensionality (Central Limit Theorem), it is obvious that, at a certain number of dimensions,  $l_{max}$  becomes smaller than  $E[nn^{dist}]$  (given by equation (7)). In that case, if we enlarge the MBR by the expected NN-distance according to Minkowski sum, the resulting region covers the entire data space. The probability of visiting a block is 1. Consequently, all blocks must be accessed, and even NN-search by an optimal search algorithm degrades to a (poor) scan of the entire data set. In Figure 7,

<sup>2</sup>Actually, it is not optimal to split in the middle as is shown in [6]. However, an unbalanced splitting strategy also fits our general case discussed in section 3.4.

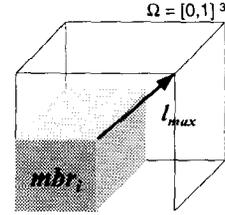


Figure 6:  $l_{max}$  in  $\Omega = [0, 1]^3$  and  $d' = 2$

$l_{max}$  (for varying  $d'$ ) is overlaid on the plot of the expected NN-distance, as a function of dimensionality ( $m = 100$ ). For these configurations, if the dimensionality exceeds around 60, then the entire data set must be accessed, even for very large databases.

### 3.3 Data-Partitioning Methods

Data-partitioning methods like R-tree, X-tree and M-tree partition the data space hierarchically in order to reduce the search cost from  $O(N)$  to  $O(\log(N))$ . Next, we investigate such methods first with rectangular, and then with spherical MBRs. In both cases, we establish lower bounds on their average search costs. Our goal is to show the (im)practicability of existing methods for NN-search in HDVSSs. In particular, we show that a sequential scan out-performs these more sophisticated hierarchical methods, even at relatively low dimensionality.

#### 3.3.1 Rectangular MBRs

Index methods such as R\*-tree [2], X-tree [7] and SR-tree [24]<sup>3</sup> use hyper-cubes to bound the region of a block. Usually, splitting a node results in two new, equally-full partitions of the data space. As discussed in Section 3.2, only  $d' < d$  dimensions are split at high dimensionality (see equation (10)), and, thus, the rectangular MBR has  $d'$  sides with a length of  $\frac{1}{2}$ , and  $d - d'$  sides with a length of 1. Following our general cost model, the probability of visiting a block during NN-search is given by the volume of that part of the extended box that lies within the data space. Figure 8 shows the probability of accessing a block during a NN-search for different database sizes, and different values of  $d'$ . The graphs are only plotted for dimensions above the number of split axes, that is 10, 14 and 17 for  $10^5$ ,  $10^6$  and  $10^7$  data points, respectively. Depending upon the database size, the 20% threshold is exceeded for dimensionality greater than around  $d = 15$ ,  $d = 18$ , and  $d = 20$ . Based on our earlier assumption about the performance of scan algorithms, these values provide upper bounds on the dimensionality at which any data-partitioning method with rectangular MBRs can be expected to perform 'well'. On the other hand, for low-dimensional spaces (that is, for  $d < 10$ ), there is considerable scope for data-partitioning methods to be

<sup>3</sup>SR-tree also uses hyper-spheres.

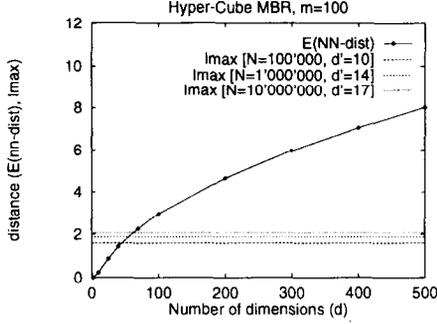


Figure 7: Comparison of  $l_{max}$  with  $E[nn^{dist}]$ .

effective in pruning the search space for efficient NN-search (as is well-known in practice).

### 3.3.2 Spherical MBRs

The analysis above applies to index methods whose MBRs are hyper-cubes. There exists another group of index structures, however, such as the TV-tree [25], M-tree [10] and SR-tree [24], which use MBRs in the form of hyper-spheres. In an optimal structure, each block consists of the center point  $C$  and its  $m - 1$  nearest neighbors (where  $m$  again denotes the average number of data points per block). Therefore, the MBR can be described by the NN-sphere  $nn^{sp,m-1}(C)$  whose radius is given by  $nn^{dist,m-1}(C)$ . If we now use a Minkowski sum to transform this region, we enlarge the MBR by the expected NN-distance  $E[nn^{dist}]$ . The result is a new hyper-sphere given by

$$sp^d(C, nn^{dist,m-1}(C) + E[nn^{dist}])$$

The probability, that block  $i$  must be visited during a NN-search can be formulated as:

$$P_{visit}^{sp}[i] \geq Vol(sp^d(C, nn^{dist,m-1}(C) + E[nn^{dist}]) \cap \Omega)$$

Since  $nn^{dist,i}$  does not decrease as  $i$  increases (that is,  $\forall j > i : nn^{dist,j} \geq nn^{dist,i}$ ), another lower bound for this probability can be obtained by replacing  $nn^{dist,m-1}$  by  $nn^{dist,1} = E[nn^{dist}]$ :

$$P_{visit}^{sp}[i] \geq Vol(sp^d(C, 2 \cdot E[nn^{dist}]) \cap \Omega) \quad (12)$$

In order to obtain the probability of accessing a block during the search, we average the above probability over all center points  $C \in \Omega$ :

$$P_{visit}^{sp,avg} \geq \int_{C \in \Omega} Vol(sp^d(C, 2 \cdot E[nn^{dist}]) \cap \Omega) dC \quad (13)$$

Figure 9 shows that the percentage of blocks visited increases rapidly with the dimensionality, and reaches 100% with  $d = 45$ . This is a similar pattern to that observed above for hyper-cube MBRs. For  $d = 26$ , the critical performance threshold of 20% is already exceeded, and a sequential scan will perform better in practice, on average.

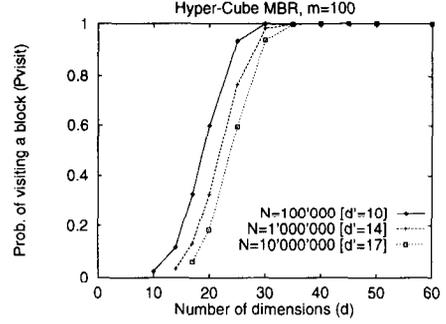


Figure 8: Probability of accessing a block with rectangular MBRs.

### 3.4 General Partitioning and Clustering Schemes

The two preceding sections have shown that the performance of many well-known index methods degrade with increased dimensionality. In this section, we now show that no partitioning or clustering scheme can offer efficient NN-search if the number of dimensions becomes large. In particular, we demonstrate that the complexity of such methods becomes  $O(N)$ , and that a large portion (up to 100%) of data blocks must be read in order to determine the nearest neighbor.

In this section, we do not differentiate clustering, data- and space-partitioning methods. Each of these methods collects several data points which form a partition/cluster, and stores these points in a single block. We do not consider the organization of these partitions/clusters since we are only interested in the percentage of clusters that must be accessed during NN-searches. In the following, the term ‘cluster’ denotes either a partition in a space- or data-partitioning scheme, or a cluster in a clustering scheme. In order to establish lower bounds on the probability of accessing a cluster, we need the following basic assumptions:

1. A cluster is characterized by a geometrical form (MBR) that covers all cluster points,
2. Each cluster contains at least two points, and
3. The MBR of a cluster is convex.

These basic assumptions are necessary for indexing methods in order to allow efficient pruning of the search space during NN-searches. Given a query point and the MBR of a cluster, the contents of the cluster can be excluded from the search, if and only if no point within its MBR is a candidate for the nearest neighbor of the query point. Thus, it must be possible to determine bounds on the distance between any point within the MBR and the query point. Further, assume that the second assumption were not to hold, and that only a single point is stored in a cluster, and the resulting structure is a sequential file. In a tree

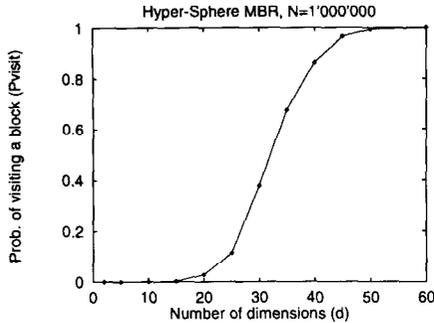


Figure 9: Probability of accessing a block with spherical MBRs.

structure, as an example, the problem is then simply shifted to the next level of the tree.

**Lower Bounds on the Probability of Accessing a Block:** Let  $l$  denote the number of clusters. Each cluster  $C_i$  is delimited by a geometrical form  $mbr(C_i)$ . Based on the general cost model, we can determine the average probability of accessing a cluster during an NN-search (the function  $VM(\bullet)$  is further used as an abbreviation of the volume of the Minkowski sum):

$$VM(x) \equiv Vol \left( MSum \left( x, E[nn^{dist}] \right) \cap \Omega \right) \quad (14)$$

$$P_{visit}^{avg} = \frac{1}{l} \sum_{i=1}^l VM(mbr(C_i))$$

Since each cluster contains at least two points, we can pick two arbitrary data points (say  $A_i$  and  $B_i$ ) out of the cluster and join them by the line  $line(A_i, B_i)$ . As  $mbr(C_i)$  is convex,  $line(A_i, B_i)$  is contained in  $mbr(C_i)$ , and, thus, we can lower bound the volume of the extended  $mbr(C_i)$  by the volume of the extension of  $line(A_i, B_i)$ :

$$VM(mbr(C_i)) \geq VM(line(A_i, B_i)) \quad (15)$$

In order to underestimate the volume of the extended lines joining  $A_i$  and  $B_i$ , we build line clusters with  $A_i$  that have an optimal (i.e. minimal) minkowski sum, and, thus, the probability of accessing these line clusters becomes minimal. The minkowski sum depends on the length and the position of the line. On average, we can lower bound the distance between  $A_i$  and  $B_i$  by the expected NN-distance (equation (7)) and the optimal line cluster (i.e. the one with the minimal minkowski sum) for point  $A_i$  is the line  $line(A_i, P_i)$ , with  $P_i \in surf(nn^{sp}(A_i))$ <sup>4</sup>, such that there is no other point  $Q \in surf(nn^{sp}(A_i))$  with a smaller minkowski sum for  $line(A_i, Q)$ :

$$\begin{aligned} VM(line(A_i, B_i)) &\geq VM(line(A_i, P_i)) \\ &= \min_{Q \in surf(nn^{sp}(A_i))} VM(line(A_i, Q)) \\ &\quad \text{with } P_i \in surf(nn^{sp}(A_i)) \end{aligned} \quad (16)$$

<sup>4</sup>  $surf(\bullet)$  denotes the surface of  $\bullet$

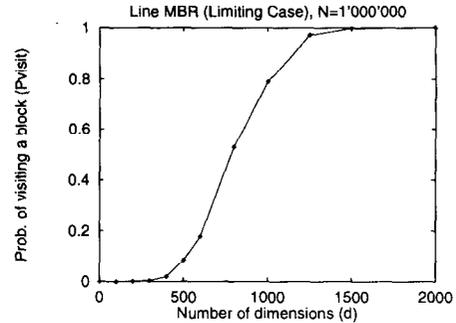


Figure 10: Probability of accessing a block in a general indexing scheme.

Equation (16) can easily be verified by assuming the contrary—i.e. the average Minkowski sum of  $line(A_i, B_i)$  is smaller than the one of  $line(A_i, P_i)$ —and deriving a contradiction.

Therefore, we can lower bound the average probability of accessing a line clusters by determining the average volume of minkowski sums over all possible pairs  $A$  and  $P(A)$  in the data space:

$$P_{visit}^{avg} = \frac{1}{l} \sum_{i=1}^l VM(mbr(C_i)) \geq \int_{A \in \Omega} VM(line(A, P(A))) dA \quad (17)$$

with  $P(A) \in surf(nn^{sp}(A))$  and minimizing the Minkowski sum analogously to equation (16). In Figure 10, this lower bound on  $P_{visit}^{avg}$  is plotted which was obtained by a Monte Carlo simulation of equation (17). The graph shows that  $P_{visit}^{avg}$  steadily increases and finally converges to 1. In other words, all clusters must be accessed in order to find the nearest neighbor. Based on our assumption about the performance of scan algorithms, the 20% threshold is exceeded when  $d \geq 610$ . In other words, no clustering or partitioning method can offer better performance, on average, than a sequential scan, at dimensionality greater than 610. From equation (17) and Figure 10 we draw the following (obvious) conclusions (always having our assumptions in mind):

**Conclusion 1 (Performance)** For any clustering and partitioning method there is a dimensionality  $\hat{d}$  beyond which a simple sequential scan performs better. Because equation (17) establishes a crude estimation, in practice this threshold  $\hat{d}$  will be well below 610.

**Conclusion 2 (Complexity)** The complexity of any clustering and partitioning methods tends towards  $O(N)$  as dimensionality increases.

**Conclusion 3 (Degeneration)** For every partitioning and clustering method there is a dimensionality  $\bar{d}$  such that, on average, all blocks are accessed if the number of dimensions exceeds  $\bar{d}$ .

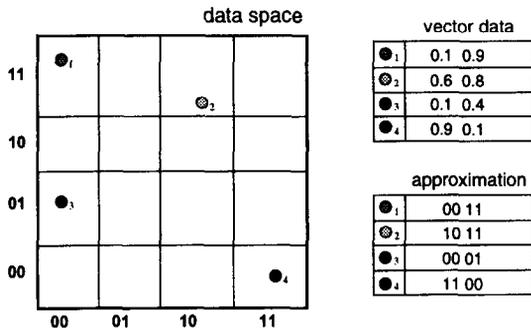


Figure 11: Building the VA-File

## 4 Object Approximations for Similarity Search

We have shown that, as dimensionality increases, the performance of partitioning index methods degrades to that of a linear scan. In this section, therefore, we describe a simple method which accelerates that unavoidable scan by using object approximations to compress the vector data. The method, the so-called *vector approximation file* (or ‘VA-File’), reduces the amount of data that must be read during similarity searches. We only sketch the method here, and present the most relevant performance measurements. The interested reader is referred to [34] for more detail.

### 4.1 The VA-File

The *vector approximation file* (VA-File) divides the data space into  $2^b$  rectangular cells where  $b$  denotes a user specified number of bits (e.g. some number of bits per dimension). Instead of hierarchically organizing these cells like in grid-files or R-trees, the VA-File allocates a unique bit-string of length  $b$  for each cell, and approximates data points that fall into a cell by that bit-string. The VA-File itself is simply an array of these compact, geometric approximations. Nearest neighbor queries are performed by *scanning* the entire approximation file, and by excluding the vast majority of vectors from the search (filtering step) based only on these approximations.

**Compressing Vector Data:** For each dimension  $i$ , a small number of bits ( $b_i$ ) is assigned ( $b_i$  is typically between 4 and 6), and  $2^{b_i}$  slices along the dimension  $i$  are determined in such a way that all slices are equally full. These slices are numbered  $0, \dots, 2^{b_i} - 1$  and are kept constant while inserting, deleting and updating data points. Let  $b$  be the sum of all  $b_i$ , i.e.  $b = \sum_{i=1}^d b_i$ . Then, the data space is divided into  $2^b$  hyper-rectangular cells, each of which can be represented by a unique bit-string of length  $b$ . Each data point is approximated by the bit-string of the cell into which it falls. Figure 11 illustrates this for five sample points. In addition to the basic vector data and the approximations, only the boundary points along each

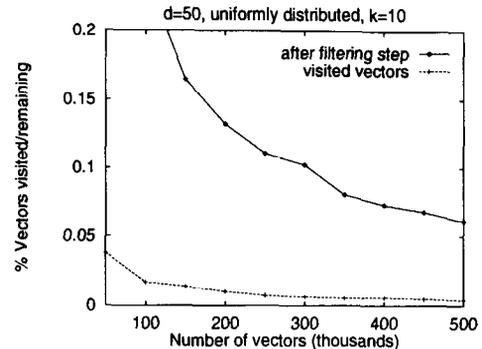


Figure 12: Vector selectivity for the VA-File as a function of the database size.  $b_i = 6$  for all experiments.

dimension must be stored. Depending upon the accuracy of the data points and the number of bits chosen, the approximation file is 4 to 8 times smaller than the vector file. Thus, storage overhead ratio is very small, on the order of 0.125 to 0.25.

Assume that for each dimension a small number of bits is allocated (i.e.  $b_i = l$ ,  $b = d \cdot l$ ,  $l = 4 \dots 8$ ), and that the slices along each dimension are of equal size. Then, the probability that a point lies within a cell is proportional to volume of the cell:

$$P[\text{"in cell"}] = Vol(\text{cell}) = \left(\frac{1}{2^{b_i}}\right)^d = 2^{-b} \quad (18)$$

Given an approximation of a vector, the probability that at least one vector shares the same approximation is given by:

$$P[\text{share}] = 1 - (1 - 2^{-b})^{N-1} \approx \frac{N}{2^b} \quad (19)$$

Assuming  $N = 10^6 \approx 2^{20}$  and  $b = 100$ , the above probability is  $2^{-80}$ , and it becomes very unlikely that several vectors lie in the same cell and share the same approximation. Further, the number of cells ( $2^b$ ) is much larger than the number of vectors ( $N$ ) such that the vast majority of the cells must be empty (compare with observation 1). Consequently, we can use rough approximations without risk of collisions. Obviously, the VA-File benefits from the sparseness of HDVS as opposed to partitioning or clustering methods.

**The Filtering Step:** When searching for the nearest neighbor, the entire approximation file is scanned and upper and lower bounds on the distance to the query can easily be determined based on the rectangular cell represented by the approximation. Assume  $\delta$  is the smallest upper bound found so far. If an approximation is encountered such that its lower bound exceeds  $\delta$ , the corresponding object can be eliminated since at least one better candidate exists. Analogously, we can define a filtering step when the  $k$  nearest neighbor must be retrieved. A critical factor of the search performance is the selectivity of this filtering step since the remaining data objects are accessed in the vector

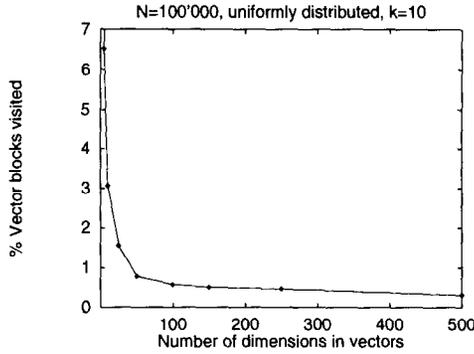


Figure 13: Block selectivity as a function of dimensionality.  $b_i = 6$  for all experiments.

file and random IO operations occur. If too many objects remain, the performance gain due to the reduced volume of approximations is lost. The selectivity experiments in Figure 12 shows improved vector selectivity as the number of data points increases ( $d = 50$ ,  $k = 10$ ,  $b_i = 6$ , uniformly distributed data). At  $N = 500'000$ , less than 0.1% ( $=500$ ) of the vectors remain after this filtering step. Note that in this figure, the range of the y-axis is from 0 to 0.2%.

**Accessing the Vectors:** After the filtering step, a small set of candidates remain. These candidates are then visited in increasing order of their lower bound on the distance to the query point  $Q$ , and the accurate distance to  $Q$  is determined. However, not all candidates must be accessed. Rather, if a lower bound is encountered that exceeds the ( $k$ -th) nearest distance seen so far, the VA-file method stops. The resulting number of accesses to the vector file is shown in Figure 12 for 10th nearest neighbor searches in a 50-dimensional, uniformly distributed data set ( $b_i = 6$ ). At  $N = 50'000$ , only 19 vectors are visited, while at  $N = 500'000$  only 20 vectors are accessed. Hence, apart of the answer set (10) only a small number of additional vectors are visited (9–10). Whereas Figure 12 plots the percentage of vectors visited, Figure 13 shows that the percentage of visited blocks in the vector file shrinks when dimensionality increases. This graph directly reflects the estimated IO cost of the VA-File and exhibits that our 20% threshold is not reached by far, even if the number of dimensions become very large. In that sense, the VA-File overcomes the difficulties of high dimensionality.

## 4.2 Performance Comparison

In order to demonstrate experimentally that our analysis is realistic, and to demonstrate that the VA-File methods is a viable alternative, we performed many evaluations based on synthetic as well as real data sets. The following four search structures were evaluated: The VA-File, the R\*-tree, the X-tree and a simple sequential scan. The synthetic data set consisted of uni-

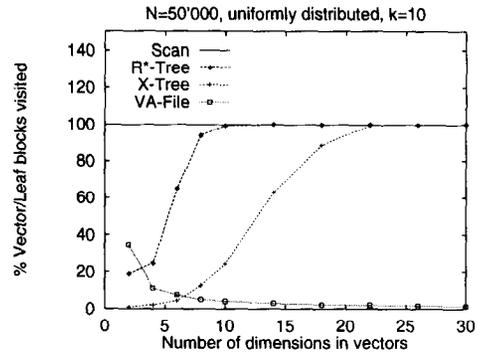


Figure 14: Block selectivity of synthetic data.  $b_i = 8$  for all experiments.

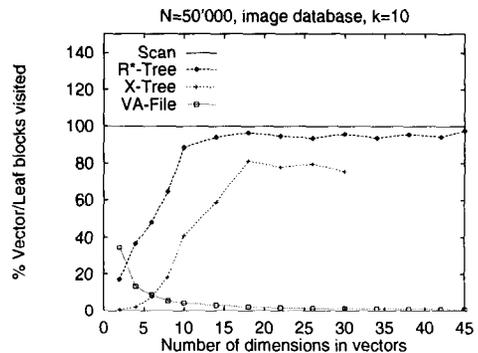


Figure 15: Block selectivity of real data.  $b_i = 8$  for all experiments.

formly distributed data points. The real data set was obtained by extracting 45-dimensional feature vectors from an image database containing more than 50'000 images<sup>5</sup>. The number of nearest neighbor to search was always 10 (i.e.  $k = 10$ ). All experiments were performed on a Sun SPARCstation 4 with 64 MBytes of main memory and all data was stored on its local disk. The scan algorithm retrieved data in blocks of 400K. The block size of the X-tree, R\*-tree and the vector file of the VA-File was always 8K. The number of bits per dimensions was 8.

Figure 14 and 15 depicts the percentage of blocks visited for the synthetic and the real data set, respectively, as a function of dimensionality. As predicted by our analysis, the tree-methods degenerate to a scan through all leaf nodes. In practice, based on our 20% threshold, the performance of these data-partitioning methods becomes worse than that of a simple scan if dimensionality exceeds 10. On the other hand, the VA-File improves with dimensionality and outperforms the

<sup>5</sup>The color similarity measure described by Stricker and Orengo [32] generates 9-dimensional feature vectors for each image. A newer approach treats five overlapping parts of images separately, and generates a 45-dimensional feature vector for each image. Note: this method is not based on color histograms.

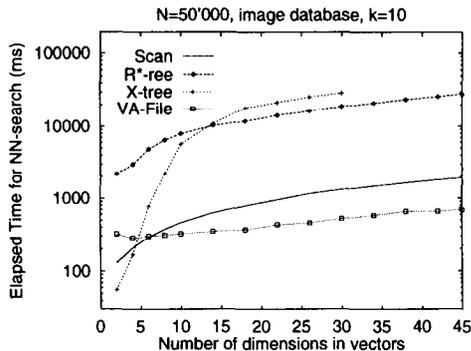


Figure 16: Wall-clock time for the image database.  $b_i = 8$  for all experiments.

tree-methods beyond a dimensionality of 6.

We further performed timing experiments based on the real data set and on a 5-dimensional, uniformly distributed data set. In Figure 16, the elapsed time for 10th nearest neighbor searches in the real data set with varying dimensionality is plotted. Notice that the scale of the  $y$ -axis is logarithmic in this figure. In low-dimensional data spaces, the sequential scan ( $5 \leq d \leq 6$ ) and the X-tree ( $d < 5$ ) produce least disk operation and execute the nearest neighbor search fastest. In high-dimensional data spaces, that is  $d \geq 6$ , the VA-File outperforms all other methods. In Figure 17, the wall-clock results for the 5-dimensional, uniformly distributed data set with growing database size is shown (the graph of R\*-tree is skipped since all values were above 2 seconds). The cost of the X-tree method grows linearly with the increasing number of points, however, the performance gain compared to the VA-File is not overwhelming. In fact, for higher dimensional vector spaces ( $d \geq 6$ ), the X-tree has lost its advantage and the VA-File performs best.

## 5 Conclusions

In this paper we have studied the impact of dimensionality on the nearest-neighbor similarity-search in HDVSs from a theoretical and practical point of view. Under the assumption of uniformity and independence, we have established lower bounds on the average performance of NN-search for space- and data-partitioning, and clustering structures. We have shown that these methods are out-performed by a simple sequential scan at moderate dimensionality (i.e.  $d = 10$ ). Further, we have shown that any partitioning scheme and clustering technique must degenerate to a sequential scan through all their blocks if the number of dimension is sufficiently large. Experiments with synthetic and real data have shown that the performance of R\*-trees and X-trees follows our analytical prediction, and that these tree-base structures are outperformed by a sequential scan by orders of magnitude if dimensionality becomes large.

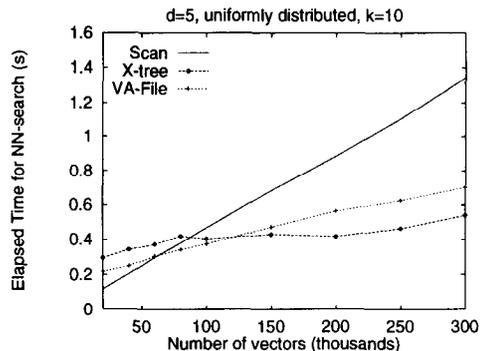


Figure 17: Wall-clock time for a uniformly distributed data set ( $d = 5$ ).  $b_i = 8$  for all experiments.

Although real data sets are not uniformly distributed and dimensions may exhibit correlations, our practical experiments have shown that multidimensional index structures are not always the most appropriate approach for NN-search. Our experiments with real data (color feature of a large image database) exhibits the same degeneration as with uniform data if the number of dimensions increases.

Given this analytical and practical basis, we postulate that all approaches to nearest-neighbor search in HDVSs ultimately become linear at high dimensionality. Consequently, we have described the VA-File, an approximation-based organization for high-dimensional data-sets, and have provided performance evaluation for this and other methods. At moderate and high dimensionality ( $d \geq 6$ ), the VA-File method can out-perform any other method known to the authors. We have also shown that performance for this method even improves as dimensionality increases.

The simple and flat structure of the VA-File also offers a number of important advantages such as parallelism, distribution, concurrency and recovery, all of which are non-trivial for hierarchical methods. Moreover, the VA-File also supports weighted search, thereby allowing relevance feedback to be incorporated. Relevance feedback can have a significant impact on search effectiveness.

We have implemented an image search engine and provide a demo version on about 10'000 images. The interested reader is encouraged to try it out.

<http://www-dbs.inf.ethz.ch/weber-cgi/chariot.cgi>

## Acknowledgments

This work has been partially funded in the framework of the European ESPRIT project HERMES (project no. 9141) by the Swiss *Bundesamt für Bildung und Wissenschaft* (BBW, grant no. 93.0135), and partially by the ETH cooperative project on *Integrated Image Analysis and Retrieval*. We thank the authors of [2, 7] for making their R\*-tree and X-tree implementations available to us. We also thank Paolo Ciaccia, Pavel Zezula, Stefan Berchtold for many helpful discussions on the subject of this paper. Finally, we thank the reviewers for their helpful comments.

## References

- [1] D. Barbara, W. DuMouchel, C. Faloutsos, P. J. Haas, J. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Data Engineering*, 20(4):3–45, 1997.
- [2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R\*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 322–331, Atlantic City, NJ, 23–25 May 1990.
- [3] J. Bentley and J. Friedman. Data structures for range searching. *ACM Computing Surveys*, 11(4):397–409, December 1979.
- [4] S. Berchtold, C. Böhm, B. Braunmüller, D. Keim, and H.-P. Kriegel. Fast parallel similarity search in multimedia databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 1–12, Tucson, USA, 1997.
- [5] S. Berchtold, C. Böhm, D. Keim, and H.-P. Kriegel. A cost model for nearest neighbour search. In *Proc. of the ACM Symposium on Principles of Database Systems*, pages 78–86, Tucson, USA, 1997.
- [6] S. Berchtold, C. Böhm, and H.-P. Kriegel. Improving the query performance of high-dimensional index structures by bulk load operations. In *Proc. of the Int. Conf. on Extending Database Technology*, volume 6, pages 216–230, Valencia, Spain, March 1998.
- [7] S. Berchtold, D. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proc. of the Int. Conference on Very Large Databases*, pages 28–39, 1996.
- [8] T. Brinkhoff, H.-P. Kriegel, and R. Schneider. Comparison of approximations of complex objects used for approximation-based query processing in spatial database systems. In *International Conference on Data Engineering*, pages 40–49, Los Alamitos, Ca., USA, Apr. 1993.
- [9] T. Brinkhoff, H.-P. Kriegel, R. Schneider, and B. Seeger. Multi-step processing of spatial joins. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 23(2):197–208, June 1994.
- [10] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proc. of the Int. Conference on Very Large Databases*, Athens, Greece, 1997.
- [11] J. Cleary. Analysis of an algorithm for finding nearest-neighbors in euclidean space. *ACM Transactions on Mathematical Software*, 5(2), 1979.
- [12] A. Csillaghy. *Information extraction by local density analysis: A contribution to content-based management of scientific data*. Ph.D. thesis, Institut für Informationssysteme, 1997.
- [13] A. Dimai. Differences of global features for region indexing. Technical Report 177, ETH Zürich, Feb. 1997.
- [14] C. Faloutsos. Access methods for text. *ACM Computing Surveys*, 17(1):49–74, Mar. 1985. Also published in/as: “Multiattribute Hashing Using Gray Codes”, ACM SIGMOD, 1986.
- [15] C. Faloutsos. *Searching Multimedia Databases By Content*. Kluwer Academic Press, 1996.
- [16] C. Faloutsos and S. Christodoulakis. Description and performance analysis of signature file methods for office filing. *ACM Transactions on Office Information Systems*, 5(3):237–257, July 1987.
- [17] C. Faloutsos and I. Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proc. of the ACM Symposium on Principles of Database Systems*, 1994.
- [18] R. Finkel and J. Bentley. Quad-trees: A data structure for retrieval on composite keys. *ACTA Informatica*, 4(1):1–9, 1974.
- [19] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The QBIC system. *Computer*, 28(9):23–32, Sept. 1995.
- [20] J. Friedman, J. Bentley, and R. Finkel. An algorithm for finding best-matches in logarithmic time. *TOMS*, 3(3), 1977.
- [21] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, Boston, MA, June 1984.
- [22] J. Hellerstein, E. Koutsoupias, and C. Papadimitriou. On the analysis of indexing schemes. In *Proc. of the ACM Symposium on Principles of Database Systems*, 1997.
- [23] G. Hjaltason and H. Samet. Ranking in spatial databases. In *Proceedings of the Fourth International Symposium on Advances in Spatial Database Systems (SSD95)*, number 951 in Lecture Notes in Computer Science, pages 83–95, Portland, Maine, Aug. 1995. Springer Verlag.
- [24] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, Tucson, Arizona USA, 1997.
- [25] K.-I. Lin, H. Jagadish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *The VLDB Journal*, 3(4):517–549, Oct. 1994.
- [26] D. Lomet. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Transactions on Database Systems*, 15(4):625–658, December 1990.
- [27] J. Nievergelt, H. Hinterberger, and K. Sevcik. The grid file: An adaptable symmetric multikey file structure. *ACM Transactions on Database Systems*, 9(1):38–71, Mar. 1984.
- [28] J. Robinson. The k-d-b-tree: A search structure for large multidimensional dynamic indexes. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 10–18, 1981.
- [29] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.
- [30] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R+-tree: A dynamic index for multi-dimensional objects. In *Proc. of the Int. Conference on Very Large Databases*, pages 507–518, Brighton, England, 1987.
- [31] R. Sproull. Refinements to nearest-neighbor search in k-dimensional trees. *Algorithmica*, 1991.
- [32] M. Stricker and M. Orengo. Similarity of color images. In *Storage and Retrieval for Image and Video Databases, SPIE*, San Jose, CA, 1995.
- [33] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 1. Computer Science Press, 1988.
- [34] R. Weber and S. Blott. An approximation based data structure for similarity search. Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997. Available at <http://www-dbs.ethz.ch/~weber/paper/TR1997b.ps>.