Predicting the Performance of Index Structures for High-Dimensional Datasets

Christian A. Lang
clang@cs.ucsb.edu

Ambuj K. Singh
ambuj@cs.ucsb.edu

Technical Report TRCS99-34
Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

We present two new models for predicting the number of index page accesses during nearest neighbor queries for high-dimensional datasets, a *density-based model* that depends weakly on the underlying index structure and results in coarser predictions, and a *sampling-based model* that depends strongly on the underlying index structure and gives more accurate predictions. We give detailed evaluations of both models and validate them experimentally by considering a number of synthetic and real datasets. We analyze the error in the models and illustrate how it can be minimized. As an application of our models, we show how the sampling-based model can be used to determine the optimal number of index dimensions for a high-dimensional dataset. Since existing models are tailored either to uniform datasets or to a small number of dimensions, this research makes a significant contribution to the understanding of high-dimensional multimedia data.

1 Introduction

Rapid advances in information technology have opened up new application areas for the growing field of multi-media databases. User interface and access to multi-media objects is typically provided through similarity search which finds objects close to a query object. The distance calculations are typically performed on abstractions of the objects, called *feature vectors*. The number of dimensions extracted from a multi-media object in order to obtain an accurate representation can become quite high. For example, for finding a similar audio, more than 20 dimensions are commonly used [WBKW96]. The ImageRover system uses 768 dimensions for finding a similar image based on color and texture [STC97]. Even though the number of dimensions can be reduced through transformations such as SVD and FFT, dimensionality over 50 is quite common in image and video databases.

Several indexing techniques have also been proposed for searching in high-dimensional Euclidean spaces [BKSS90, BKK96, Gut84, KS97, WJ96b, LS90, Fre87, Fre95, Fre96, LJF94, CM99, Ben75, BBK98, WSB98]. They all have advantages and disadvantages depending on the underlying dataset, the number of dimensions, and the query type. Recently, an efficient index structure called M-tree [Ozs97] has been proposed for general metric distances. The generality however comes at the price of a worse performance.

We consider the problem of predicting the performance of index structures for nearest neighbor (NN) queries in Euclidean space. This problem is important for a number of reasons. First, given the plethora of index structures, it is not obvious which index structure is the best for a given dataset. It would be helpful if there was a quick way to determine this. Second, even for a fixed index structure and a given dataset, there is a flexibility in the number of dimensions to use in building the index structure. An index structure on a reduced number of dimensions can be used with a two-step searching [KSF$^+$96, SK98] involving all dimensions to speed up the search. However, the choice of the optimal number of reduced dimensions is far from obvious. A prediction model such as that proposed in this paper can help in reaching this decision. Finally, a prediction model can also help in query planning, in understanding the behavior of index structures in high-dimensional spaces, and in the derivation of new and more efficient index structures.

Modeling the behavior of index structures on real datasets is complex due to a number of reasons. First, the page extent and page layout of real datasets is non-uniform and difficult to predict in high dimensions. Second, the query spheres are non-uniform due to the clustering of data. As a result, pages have different access probabilities and obtaining the average number of pages accessed by a query can be challenging. Previous work on performance prediction has considered either a small number of dimensions [FK94, PM97], or the case of uniform data distributions [BBKK97, WSB98].

We present two models for predicting the performance of index structures. The first model estimates the average density in the dataspace to predict the page extents and accesses. The second model bases the prediction on a miniature version of the index built in memory. Both models first predict the layout of the index pages and then compute the average number of accesses to these pages. We assume that the query distribution follows the data distribution. For each model, we analyze and show how to compensate for the error. In order to validate our models, we used a number of synthetic and real datasets. These are shown in Table 1. The TEXTURE and COLOR datasets are transformed using the Karhunen-Loève transform.

UNIFORM64: 100,000 uniformly distributed 64-dimensional points. The dataspace is the unit hypercube. The points are located only in the $[0.45, 0.55]^d$-hypercube.
CLUSTERS64: Two clusters of uniformly distributed points (50,000 each). The dataspace is the unit hypercube. One cluster is in the $[0.0, 0.1]^d$-hypercube, the other one in the $[0.9, 1.0]^d$-hypercube.
GAUSSIAN64: 100,000 64-dimensional points in normal distribution with a mean of 0.5 and a standard deviation of 0.05 in every dimension.
COLOR64: 112,361 64-dimensional vectors representing color histograms computed from a commercial CD-ROM.
COLOR112: 112,682 112-dimensional vectors representing color histograms computed from a commercial CD-ROM.
TEXTURE48: 26,697 48-dimensional vectors representing texture feature vectors obtained from the Corel Image Collection.
TEXTURE60: 275,465 60-dimensional vectors representing texture feature vectors of Landsat images.

Table 1: Datasets used

The rest of the paper is organized as follows. Section 2 discusses related work on predicting the performance of index structures. Section 3 discusses the density-based model and Section 4 the sampling-based model. Sections 5 and 6 present some applications of the performance prediction model, specifically choosing the optimal number of reduced dimensions and the optimal index structure. We end with some concluding remarks in Section 7.

2 Related Work

The first investigation in predicting the performance of R-trees was performed by Faloutsos and Kamel [FK94]. They provide a model to predict the number of page accesses during range queries. Their model is based on the so-called *fractal dimension* of a dataset, which can be seen as the inherent dimensionality of the dataset. Given the fractal dimension d, the effective page capacity C_{eff}, and the number of leaf pages N, they compute the sidelengths of each page as $(\frac{C_{eff}}{N})^{1/d}$. The authors assume that all pages are hypercubes with exactly the same extents and that they are all aligned along a uniform grid. Both assumptions may not hold for real index structures. For example, we found that for high dimensions the pages of the VAMSplit R-tree [WJ96a] tend to have a high variance in their extents (cf. Table 2). Knowing the page extent, the

dimensions	min. extent	max. extent
1	0.01457	0.01457
2	0.03771	0.06825
4	0.05611	0.1556
8	0.05681	0.1862
16	0.04105	0.1640
32	0.01117	0.1394
64	0.0001106	0.1115

Table 2: Average minimum and maximum extent of pages vs. number of dimensions (COLOR64 dataset)

number of page intersections for a range query can be computed. The experimental results (for 2-dimensional data) show that the relative error is rarely above 10%. The computation of the fractal dimension becomes very costly for higher dimensions. Therefore, higher dimensionalities are not discussed. Different query types are also not examined.

A first approach in investigating NN queries on non-uniform datasets was taken by Papadopoulos and Manolopoulos [PM97]. Given a dataset and its *correlation fractal dimension* D_2 (which is costly to calculate), they first derive formulae for a lower and an upper bound on the NN search radius. Once D_2 is known, the average distance of a point to its NN can be computed as a function of D_2 and the population of the dataset. This number is then used as the lower bound on the search radius.[1] In order to establish the upper bound on the search radius, the authors assume that all pages are hypercubes with identical sides. This may not hold for higher dimensions. Additionally, the calculation of the upper bound assumes uniformity in the query radius distribution. For both bounds, this model computes the probability of a page access as the ratio of the expected number of points in the query area (using D_2) to N. The authors do not take boundary effects into account and approximate the query sphere by a query hypercube. Both assumptions may lead to high errors in higher dimensions.

In contrast to the previous approaches, the cost model presented by Berchtold et al. [BBKK97] is suited for NN queries in high-dimensional data spaces with uniform data distribution. The authors use this uniformity assumption to get the expected query radius. They then develop a cost formula for the expected number of page accesses based on the concept of Minkowski sums. When normalizing the data space to the unit hypercube, the sum of the volumes of all page/radius Minkowski sums corresponds to the av-

[1] The presented formula holds only if queries always land on data points. We make a similar assumption for our model.

erage number of pages intersected by the query sphere. To obtain the page layouts, the authors assume that each dimension is split at most once. This assumption may not hold if the dimensions have varying significance or if there are only a few dimensions. Furthermore, this approach does not handle the case of embedded datasets that may have a smaller dimensionality than the whole dataspace. One important contribution of this paper is the consideration of boundary effects. These effects occur whenever a query point lies close to the boundary of the dataset and they are especially significant in high dimensions.

An important work showing the limitations of NN queries in high dimensions was done by Weber et al. [WSB98]. Under the assumption of uniformity and independence of the data, they show that the expected NN distance grows monotonically with the number of dimensions. They then derive a cost model based on Minkowski sums. As in [BBKK97], the authors assume that each dimension is split at most once and that the split occurs in the middle. In contrast to [BBKK97], Weber et al. give formulae for different space- and data-partitioning methods, and even for a general partitioning scheme assuming mainly the convexity of the bounding shape.

In the following sections, we present our two models for predicting the number of pages accessed during a k-NN query.

3 Density-Based Model

This section introduces the density-based prediction model and presents experimental results for one synthetic uniform dataset and one real non-uniform dataset. This is followed by an evaluation of the approach that shows its limitations and motivates the need for the second model.

d	number of dimensions
N	number of datapoints
Ω	dataspace
$\mathcal{D} \subseteq \Omega$	dataset
C	capacity of (leaf) pages
$extent_i(c)$	extent of hypercube c in dimension i
$volume(c)$	volume of hypercube c
$points(c)$	number of points of the dataset that lie within hypercube c

Table 3: Notation used in the paper

Before going into detail of the density-based model, we give the common steps of both prediction models:

Step 1: OBTAIN (LEAF)[2] PAGE LAYOUTS.

Step 2: OBTAIN k-NN QUERY RADII FOR A SMALL QUERY SAMPLE (≈ 500) OF $\mathcal{D}$ (SEE TABLE 3 FOR AN EXPLANATION OF THE NOTATION).

Step 3: COMPUTE THE AVERAGE NUMBER OF INTERSECTIONS OF EACH PAGE FROM STEP 1 WITH ALL QUERY SPHERES FROM STEP 2.

We assume that the queries follow the data distribution and each point of the dataset has the same probability of being chosen as a query point. We compute the exact query sphere for each point in the query sample. Most prediction techniques in the literature use an average search radius for the whole dataset. The calculation of several search radii at different points of the dataset leads to more accurate predictions. This is because boundary effects are automatically taken into account, since queries close to the boundary will lead to larger query spheres.

3.1 The Density-Based Prediction Algorithm

The assumption behind the density-based model is that the knowledge of the density of $\mathcal{D}$ in a small neighborhood of a point is enough to predict the layout of the page containing this point. This is a reasonable assumption since in areas of high density we expect to find small pages and in areas of low density we expect to find large pages.

Our algorithm attempts to predict a set of pages around randomly picked points of $\mathcal{D}$. To do so, it first calculates the density D around each of these points by determining a small hypercube c around the point and dividing the number of points in c by its volume:

$$D = \frac{points(c)}{volume(c)} \tag{1}$$

If we assume that the points are distributed uniformly within a page, the same density should be observed there. Therefore we have

$$D = \frac{C}{volume(page)} \tag{2}$$

To calculate the volume of a page we need to know its extents. Pages of an index structure are formed by splitting the extents of the dataspace. The dimensions split during this process are called *split dimensions*. For high dimensional datasets with a small number of points, the number of split dimensions can be well below the number of overall dimensions. This results in pages with extents smaller than the dataspace extent in the split dimensions

[2]We consider only leaf pages since it can be assumed that the inner tree pages are stored in memory and do not contribute to the overall I/O cost.

and extents equal to the dataspace extent in the non-split dimensions. To isolate the split dimensions, we take the characteristics of the underlying index structure into account. For example, the VAMSplit R-tree [WJ96a] splits the dataspace according to datapoint variance. So, the split dimensions are those with the highest variance.

In other words, if d is greater than the number of split dimensions s, some dimensions of the page will have the full extent of the dataset. Otherwise, no extent will have the full extent. In both cases let e denote the extent in the split dimensions. We have as the volume of the page

$$volume(page) = \begin{cases} e^s * \prod_{i=s+1}^{d} extent_i(\mathcal{D}) & \text{if } d > s \\ e^d & \text{if } d \leq s \end{cases} \tag{3}$$

By substituting the volume of a page in Equation 2 and equating the values of D in Equations 1 and 2, we obtain

$$e = \begin{cases} \sqrt[s]{\frac{C*volume(c)}{\prod_{i=s+1}^{d} extent_i(\mathcal{D})*points(c)}} & \text{if } d > s \\ \\ \sqrt[d]{\frac{C*volume(c)}{points(c)}} & \text{if } d \leq s \end{cases} \tag{4}$$

This extent e is then used to create a page with the original point as centroid and e as the page extent in the split dimensions and the extents of $\mathcal{D}$ in the remaining dimensions. To avoid boundary effects, we move each page until it totally lies inside Ω.

The complete algorithm is as follows:

Step 1: DETERMINE THE NUMBER OF SPLIT DIMENSIONS $s = \log_2(\frac{N}{C})$.

Step 2: PICK A NUMBER OF POINTS $q_1, \ldots, q_n$ FROM $\mathcal{D}$ AND PERFORM A k-NN QUERY WITH SPHERE $s_1, \ldots, s_n$ ON EACH OF THEM.

Step 3: PICK A NUMBER OF POINTS $p_1, \ldots, p_m$ FROM $\mathcal{D}$ AND DETERMINE A SMALL HYPERCUBE $c_1, \ldots, c_m$ AROUND EACH POINT.

Step 4: COMPUTE A PAGE FOR EACH p_i:

1. THE PAGE EXTENT IN THE SPLIT DIMENSIONS IS GIVEN BY EQUATION 4.
2. CREATE PAGE P_i WITH p_i AS CENTROID AND e AS PAGE EXTENT IN THE SPLIT DIMENSIONS AND THE EXTENTS OF $\mathcal{D}$ IN THE REMAINING DIMENSIONS.

Step 5: COMPUTE AVERAGE NUMBER OF INTERSECTIONS x_i BETWEEN EACH SPHERE s_i WITH $P_1, \ldots, P_m$.

Step 6: REPORT AVERAGE OF THE VALUES $\frac{x_i}{m} * \frac{N}{C}$ AS THE AVERAGE NUMBER OF PAGE INTERSECTIONS.

The hypercubes in Step 3 have to be small to preserve the locality of our density estimation but they should also not be too small or we cannot obtain good estimates. In our experiments we chose identical points for q_i and p_i and used the bounding box around the query sphere s_i as the hypercube c_i. The calculation overhead is reduced to calculating n k-NNs.

3.2 Time and Space Requirements

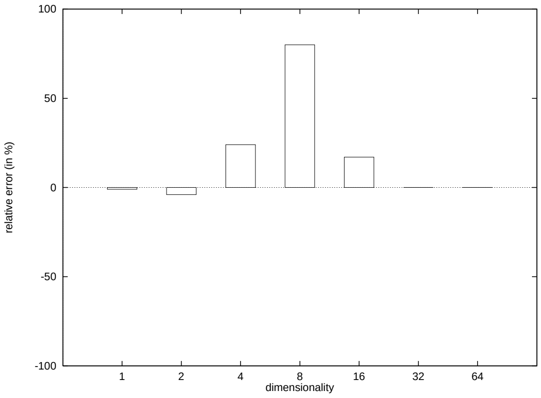
The time required for the prediction is equivalent to three scans over the whole dataset. The first scan is needed to pick the q_i (and p_i), the second scan computes the NNs for all of them, and the third scan is used to calculate the values $points(c_i)$. The space required by the algorithm is the space required to store n radii and m hypercubes.

3.3 Experimental Results

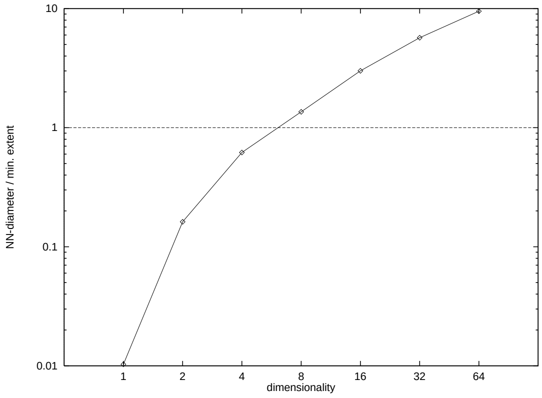
We performed experiments on two datasets: the synthetic dataset UNIFORM64, and the real dataset COLOR64. In order to understand the behavior of the model, we consider a varying number of dimensions for each dataset. The reduction was achieved by keeping only the desired number of dimensions in the datasets. We performed 21-NN queries.

In order to allow for an easy comparison, we give the relative errors of the prediction. The baseline values were obtained by running 500 21-NN queries on the datasets using a VAMSplit R-tree [WJ96a] as the index structure and counting the number of leaf page accesses.

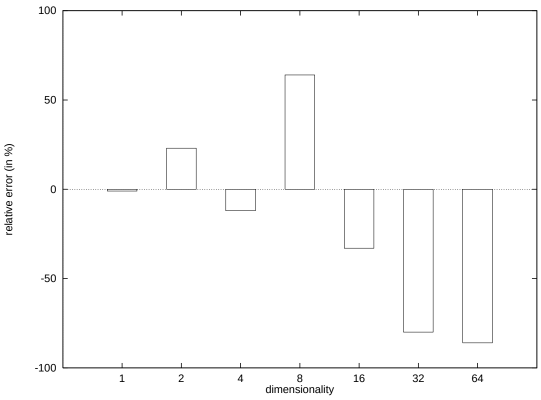
Figure 1(a) and Figure 1(c) show the relative error for different dimensions for the UNIFORM64 and the COLOR64 dataset respectively. Positive errors indicate an overestimation and negative errors indicate an underestimation. The relative error for the uniform dataset is smaller for higher dimensions, since the search radius grows monotonically to intersect all pages beyond 16 dimensions. Since we use the true NN radius at the sample queries in our prediction, all pages are observed to be accessed. Errors in predicting the page layout have little effect. At a small number of dimensions, the search radius is much smaller than the page extents. This is depicted in Figures 1(b) and 1(d) that plot the ratios of the NN diameter to the page extent. In both cases, the query sphere size reaches the page size at about 8 dimensions. As a result, for less than 8 dimensions, every page is packed with a number of sample query spheres. The estimation of the page extents using the observed densities is therefore quite accurate. The page layout for the COLOR64 dataset is much harder to predict because the pages of the index structure are not uniform, and considerable error is introduced in computing the number of accessed pages assuming uniform page extents. This drawback also existed in the earlier prediction schemes that we discussed in Section 2.



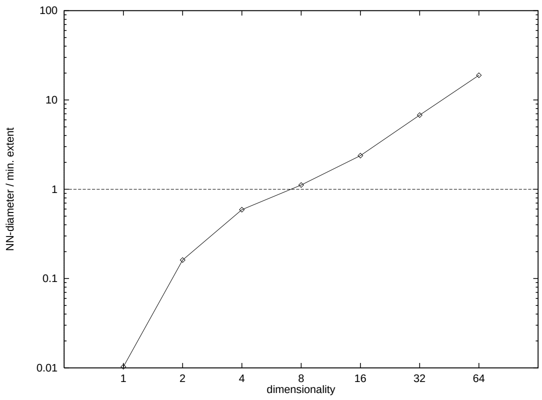
(a) UNIFORM64: relative error



(b) UNIFORM64: NN-diameter/page extent ratio



(c) COLOR64: relative error



(d) COLOR64: NN-diameter/page extent ratio

Figure 1: Relative errors for two datasets with the density-based approach

3.4 Evaluation

Our experiments show that the density-based model works well for uniform datasets and for low dimensions on non-uniform datasets. When working with uniform dataset, our model has the advantage over existing models [BBKK97] that it does not need to know the extents of uniform clusters within a larger dataspace. One drawback of the density-based model is that it assumes uniform pages. The sampling-based model discussed next frees us from this restriction.

4 Sampling-Based Model

In this section, we discuss the sampling-based prediction model. As for the density-based model, we first present the algorithm and its time and space requirements, then we present some experimental results, and finally conclude with an evaluation.

4.1 The Sampling-Based Prediction Algorithm

In the sampling-based prediction algorithm we construct a "miniature" version of the index structure in memory instead of building the full version on disk. This has the advantage that the prediction can be done very fast and can be expected to be very accurate since we use exactly the same index structure in memory whose behavior we want to predict on disks. We assume that the used index structure manages datapoints by placing pages in the dataspace. This is true for a large number of indices, like R-trees or k-d-trees.

In order to fit into memory, the dataset needs to be subsampled by a ratio ζ. To compensate for the sampling, the page capacity of the index has to be reduced by the same ratio. However, the sample pages will shrink slightly due to the removed points which will result in too few predicted page intersections. For this reason, we introduce a *compensation factor* δ by which each sample page is grown. In other words, δ is the ratio of the size of a page to the size of its corresponding sample page.

To allow for a mathematical analysis, let us assume that the points within the page are distributed uniformly. Then the problem we want to solve can be stated as follows:

Given a set of n uniformly distributed points in d dimensions. Let B be the minimal bounding hyperrectangle of these points.
How does the volume of B change when the number of points is reduced to $n * \zeta$ where $0 < \zeta < 1$?

For $d = 1$, the bounding box degenerates to an interval. Once we solve the problem for this case, we can assume the original bounding box changes the same way for each single dimension and therefore have a solution for the original problem.

The expected length $L(m)$ of the bounding interval of m uniformly distributed points whose values range from 0 to 1 can be calculated as follows: Let x_i denote the value of the ith point. There are $m!$ ways how the points can be ordered. Without loss of generality, we assume the points are ordered according to their index, i.e. $x_i < x_j$ for $i < j$. To get the expected length of the interval, we have to calculate the average of $x_k - x_1$ under all possible values for the x_i. Each x_i can only have values in the range $[0, x_{i+1})$. This leads to the following formula for the expected interval length:

$$L(m) = k! \int_0^1 \int_0^{x_m} \dots \int_0^{x_2} (x_m - x_1) dx_1 \dots dx_{m-1} dx_m \tag{5}$$

Simplifying the integrals, we get

$$L(m) = \frac{m-1}{m+1} \tag{6}$$

Therefore, if we change the number of points from n to m, the length of the interval will change on average by $\frac{L(m)}{L(n)}$ or $\frac{(m-1)(n+1)}{(m+1)(n-1)}$. So, if $m = n\zeta$, we have

$$\delta(n, \zeta) := \frac{(n\zeta - 1)(n+1)}{(n\zeta + 1)(n-1)} \tag{7}$$

as the change in interval length for n original points and a sample fraction of ζ.

Now let B' be the shrunk hyperrectangle of the original problem and let $volume(B) = e_1 \dots e_s$, where s is the number of split dimensions. Then $volume(B') = e_1\delta(n, \zeta) \dots e_s\delta(n, \zeta) = volume(B)\delta(n, \zeta)^s$.

It follows

Under uniformity assumption, the volume of each index leaf page changes by

$$\delta^{-1} = \left(\frac{(C\zeta - 1)(C + 1)}{(C\zeta + 1)(C - 1)} \right)^s$$

where C is the page capacity, ζ is the sampling fraction, and s is the number of split dimensions.

With the compensation factor derived, we can now state the whole algorithm:

sample size	NN	radius			
		2 dim	4 dim	8 dim	16 dim
100%	21	0.003581	0.02194	0.04823	0.06106
50%	21	0.003349	0.02176	0.04710	0.06061
50%	10	0.002395	0.01692	0.04048	0.05255
25%	21	0.003466	0.02087	0.04713	0.06160
25%	10	0.002372	0.01688	0.04195	0.05341
25%	5	0.001639	0.01359	0.03554	0.04676

Table 4: NN radii for different samples (COLOR64 dataset)

Step 1: PICK n SAMPLE QUERY POINTS $q_1, \dots, q_n$ FROM $\mathcal{D}$ AND PERFORM A k-NN QUERY WITH SPHERE $s_1, \dots, s_n$ ON EACH.

Step 2: SAMPLE $N * \zeta$ POINTS OF $\mathcal{D}$.

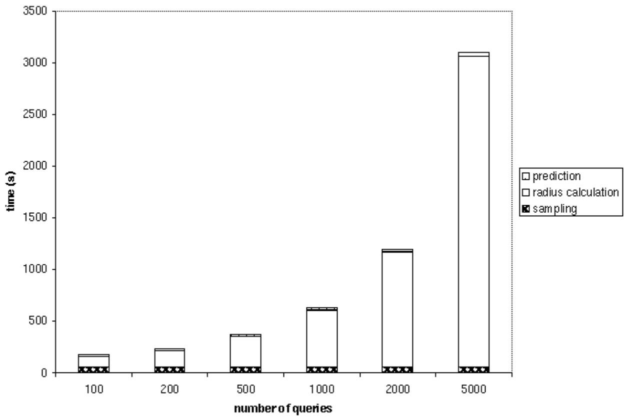
Step 3: BUILD A MINIATURE INDEX STRUCTURE IN MEMORY FOR THE SAMPLE DATASET USING $C * \zeta$ AS LEAF PAGE CAPACITY AND GROW EACH LEAF PAGE BY δ.

Step 4: COMPUTE AVERAGE NUMBER OF INTERSECTIONS x_i BETWEEN SPHERE s_i AND LEAF PAGES $P_1, \dots, P_m$ OF IN-MEMORY INDEX.

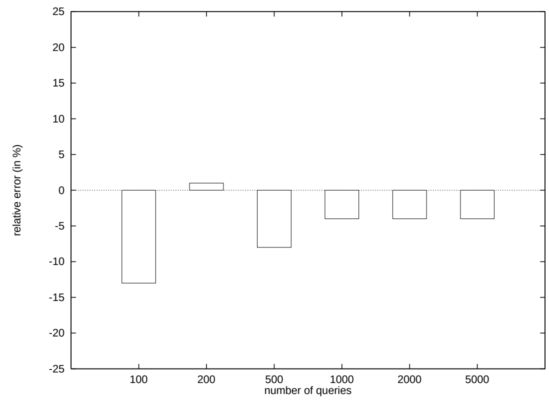
Step 5: REPORT AVERAGE OF THE VALUES $\frac{x_i}{m}$ AS EXPECTED NUMBER OF PAGE ACCESSES.

It can be expected that the resulting pages of the sample dataset have a similar layout to the disk-based index pages if the sample data points are chosen randomly. Since we compute the NN radius for the query points using the full dataset in Step 1 and compensate for the change in page size in Step 3, the intersection count in Step 4 should result in numbers similar to the disk-based index.

Of course, another option would have been to use the sample dataset itself to estimate the query spheres. (This would have eliminated Step 1 saving on one scan of the dataset.) It turns out, however, that it is difficult to relate the search radius on the sample to the search radius on the whole dataset. For example, we may expect that a 21-NN query on the whole dataset should be similar to a 10-NN query on a 50% sample and a 5-NN query on a 25% sample. As seen in Table 4, this is not true. The search radius appears to be very sensitive to the number of nearest neighbors. The 21-nn search radius on the full dataset (of 2 dimensions) is 0.003581 as compared to the 10-nn radius of 0.002395 at 50% sampling and the 5-nn radius of 0.001639 at 25% sampling. Since our prediction model depends so heavily on the computed radii, it is much better to compute the actual search radii using the whole dataset. The cost of one additional scan of the whole dataset may be worth it.



(a) Execution times



(b) Prediction error

Figure 2: Execution time and relative error for different query sizes

4.2 Time and Space Requirements

The total running time of the prediction algorithm is dominated by Step 1. We found that the total running time is almost linear in the number of query points. Even though a larger number of queries may lead to a higher accuracy, the prediction error is nearly unaffected beyond about 500 queries. These results are depicted in Figures 2(a) and 2(b). Accordingly, we choose 500 query points in all our experiments. Note that for the relative error plots, the y axis ranges now from -25% to $+25\%$.

Another important parameter of our algorithm is ζ. If we had an unbounded amount of main memory, we could always choose a 100% sample. But in general, real datasets do not fit into memory. We have a tradeoff between memory consumption and the prediction quality. To understand this tradeoff, we computed the relative error for different sample sizes for the COLOR64 dataset. The results are shown in Figure 4. As expected, the errors are higher for smaller samples (the error values are ordered left to right for 50%, 25%, 12.5%, and 6.25% samples). This effect is more pronounced for higher-dimensional data. The running time increases marginally with higher sample sizes and the memory consumption increases linearly with the sample size (cf. Figure 3(a) and 3(b)). We conclude that the largest sample size that will fit in the memory should be chosen. In all our experiments, we choose a 25% sample.

4.3 Experimental Results

Our experiments were performed on a 350-MHz Pentium-II-PC running Solaris. The programs were implemented in Java. For each dataset, we ran 500 21-NN queries on the full dataset to count the actually accessed leaf pages.

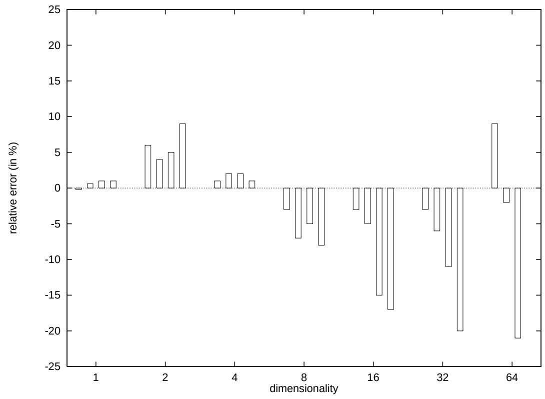
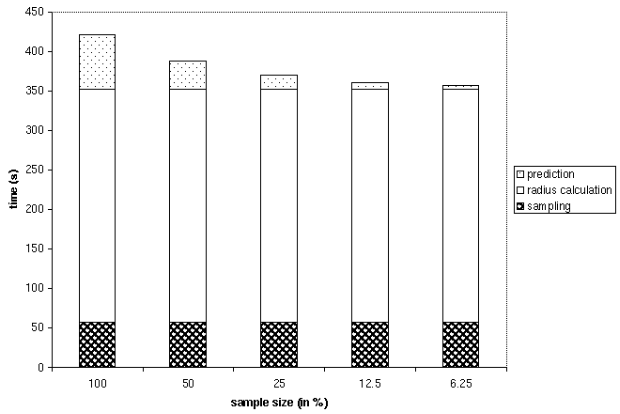


Figure 4: Relative errors for different dimensionalities (COLOR64)

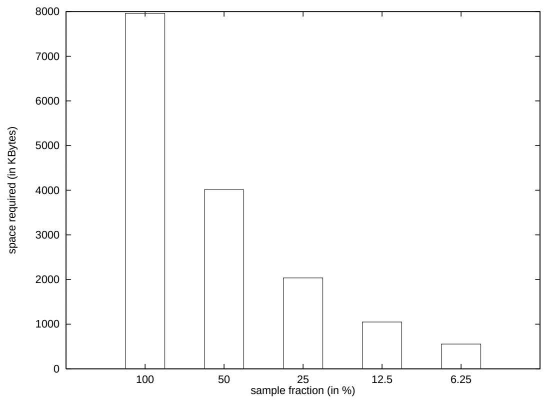
These are then compared with the prediction on a 25%-sample of the same dataset to yield a relative error. The following paragraphs present the relative errors for each dataset at different dimensions.

UNIFORM64 The relative errors for different numbers of dimensions in the dataset are shown in Figure 5. The errors are all below 10%. For higher dimensionalities (namely at 32 and 64 dimensions), the error is nearly 0. As observed earlier, this is because nearly all index pages on the whole dataset as well as on the sample dataset are accessed by each query. The model of Berchtold et al. [BBKK97] can not be applied here unless the exact layout of $\mathcal{D}$ within Ω is known.

CLUSTERS64 This dataset is different from UNIFORM64 in that it contains two uniform clusters instead



(a) Execution times



(b) Space required

Figure 3: Execution time and required space for different sample sizes

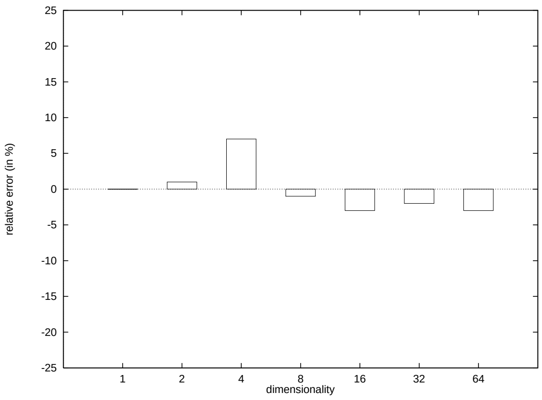


Figure 5: Relative errors for different dimensionalities (UNIFORM64, 25% sample)

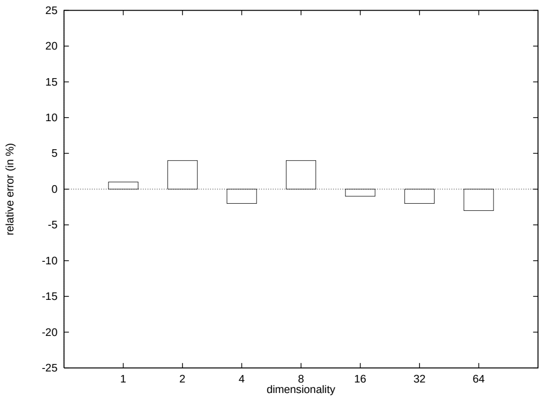


Figure 6: Relative errors for different dimensionalities (CLUSTERS64, 25% sample)

of one. Both clusters are located in opposite corners of the dataspace and small enough to not overlap each other. This dataset has the same number of points as UNIFORM64, as a result we get the same number of pages. The relative errors are depicted in Figure 6. In terms of the actual number of pages accessed, one would expect that on average about half as many pages as for the UNIFORM64 dataset are accessed. Our experiments revealed, however, that this is only true for dimensions higher than 8. For smaller dimensions, the index structure generates pages across cluster boundaries resulting in roughly the same number of pages as in the UNIFORM64 case. Our prediction algorithm yields less than 5% error for all dimensionalities, since the anomalies are also mirrored in the in-memory index. Even though this dataset is a simple extension of the UNIFORM64 dataset, none of the existing models can be used to predict the query behavior accurately.

GAUSSIAN64 This is another example of a synthetic dataset. It contains points distributed normally — a distribution we found in many real datasets, like for example the Landsat feature vectors. We generated 100,000 points with a mean of 0.5 (the center of Ω) and a standard deviation of 0.05. With these parameters, most points will lie in the hypercube $[0.4, 0.6]^d$. Now we have no longer equal densities everywhere in the cube, but a higher density in the center and lower densities near the boundary. The predictions obtained from our algorithm are shown in Figure 7. We obtained well below 5% error for all dimensionalities except 4. Here, the error was about 7%.

COLOR64 This is the first real dataset we discuss here. As opposed to the synthetic datasets considered earlier,

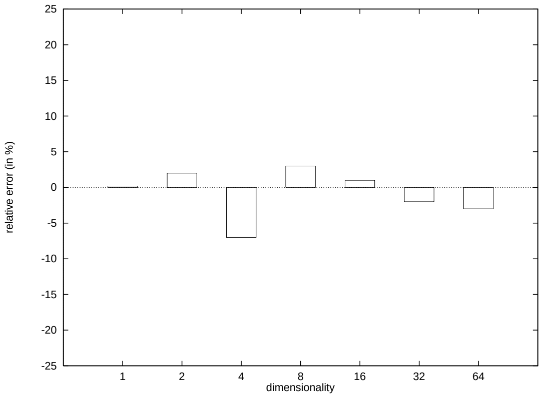


Figure 7: Relative errors for different dimensionalities (GAUSSIAN64, 25% sample)

the search radii are much smaller and only a subset of the pages is accessed even at high dimensions. Our experimental results are shown in Figure 8. As can be seen,

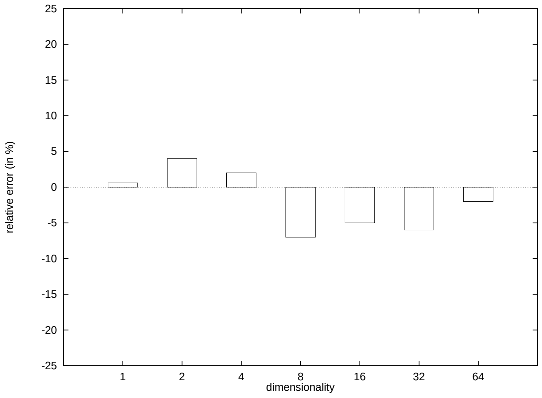


Figure 8: Relative errors for different dimensionalities (COLOR64, 25% sample)

the prediction works well even for higher dimensionalities when the clustering becomes more pronounced.

COLOR112 The overall number of pages for this dataset is higher, but the behavior of this dataset is similar to the COLOR64 dataset. The relative errors are shown in Figure 9. Relative errors are low for a small number of dimensions but for higher dimensions, our prediction deviates from the measured value by up to 18%. This can be explained by the increased deviation of the size of the sample pages from the size of the full pages. Our compensation discussed earlier assumes uniform data distribution. Since COLOR112 contains clustered data, the pages shrink faster due to sampling resulting in too small pages after the compensation in Step 3. This results in fewer page intersections in Step 4 and a higher prediction error.

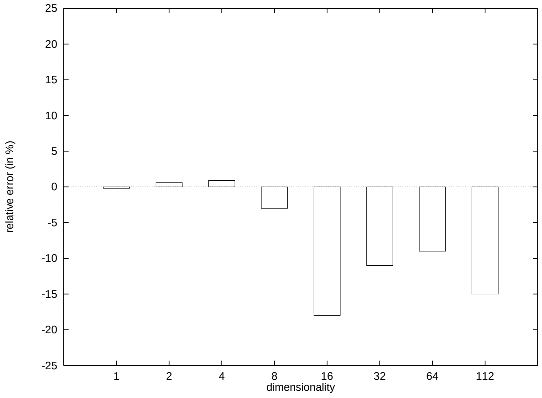


Figure 9: Relative errors for different dimensionalities (COLOR112, 25% sample)

TEXTURE60 This is the largest dataset we used in our experiments. Figure 10 shows the relative error of the predictions versus the measurement. The relative errors

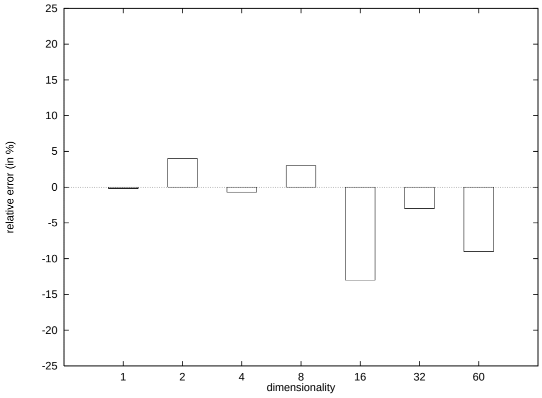


Figure 10: Relative errors for different dimensionalities (TEXTURE60, 25% sample)

for this dataset are quite low (less than 10% even at 60 dimensions).

4.4 Evaluation

The number of accessed pages depends on the search radius and the page extents. We compute the search radius exactly by considering the whole dataset. So, any error in our prediction is due to the difference between the miniature in-memory page extents and the actual disk-based page extents. We also observed that in all the experiments, the model tended to underestimate the number of page intersections for non-uniform datasets. This is due to our assumption of uniform distribution when compensating for the page volume reductions; other distributions such as Gaussian may lead to better results.

Limitations of our Approach When the number of dimensions is very high and the sample is small, the page capacity of the in-memory index may drop below 4. The R*-tree index can no longer be used. But even when using a different index structure, the page capacity eventually reaches 1 at which point each page degenerates to a point.

For example, given a dataset with one million points in 128 dimensions and a main memory of 128 Megabytes. The whole dataset has a size of about 500 Megabytes. In order to load it into memory, we have to sample it by about 20%. The leaf page capacity at 128 dimensions is about 15 and therefore the capacity of the in-memory index drops to 3.

To overcome this problem, one could try to extrapolate from lower dimensions. However, as the measured number of leaf pages for a non-SVD-transformed dataset in Figure 11 shows, simple linear extrapolation does not work.

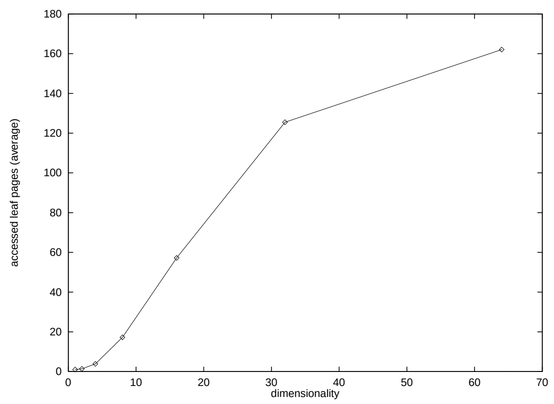


Figure 11: Average leaf page accesses vs. dimensionality (TEXTURE48, non-SVD-transformed)

5 Application 1: Determining Optimal Index Dimensionality

As an application of our approach we will show in this section how to use the sampling-based prediction model in order to determine the optimal dimensionality to build an index.

5.1 Nearest Neighbor Searching using Reduced Dimensions

Instead of building an index structure on all the dimensions, it is possible to build the index structure on a reduced number of dimensions and do a two-step searching [KSF+96, SK98] in order to obtain the NNs for a given query point. We adopt the optimal search algorithm by Seidl et al. [SK98] in this paper.

The original data space with all the dimensions is called *object* space and the reduced data space is called *feature* space. The algorithm itself maintains an index on the feature dimensions and an object server which stores the remaining dimensions.

5.2 Computing the number of accessed pages

The search algorithm presented in [SK98] ensures optimality with respect to a given feature space. Our goal in this paper is to decide which feature space is optimal with regard to the number of accessed disk pages. Examining the search algorithm presented earlier, the total number of accessed pages can be broken down into two categories: the *feature pages* accessed by the index structure, and the *object pages* accessed by the object server.

5.2.1 Computing the number of accessed feature pages

In the following, we restrict ourselves to the prediction of leaf level feature pages. This seems to be reasonable since the inner nodes of an index tree can be kept in memory in general.

In order to predict the number of accessed feature pages, we use the algorithm presented in Section 4.1 with a small modification. We obtain the search radius from the full dimension dataset but the in-memory index structure is computed for the reduced dimension dataset. This is correct since the search radius is obtained from the object server (which stores the points in all dimensions), whereas the index is built on the reduced dimensional points. The algorithm in Section 4.1 is a special case where the number of dimensions equals the number of reduced dimensions.

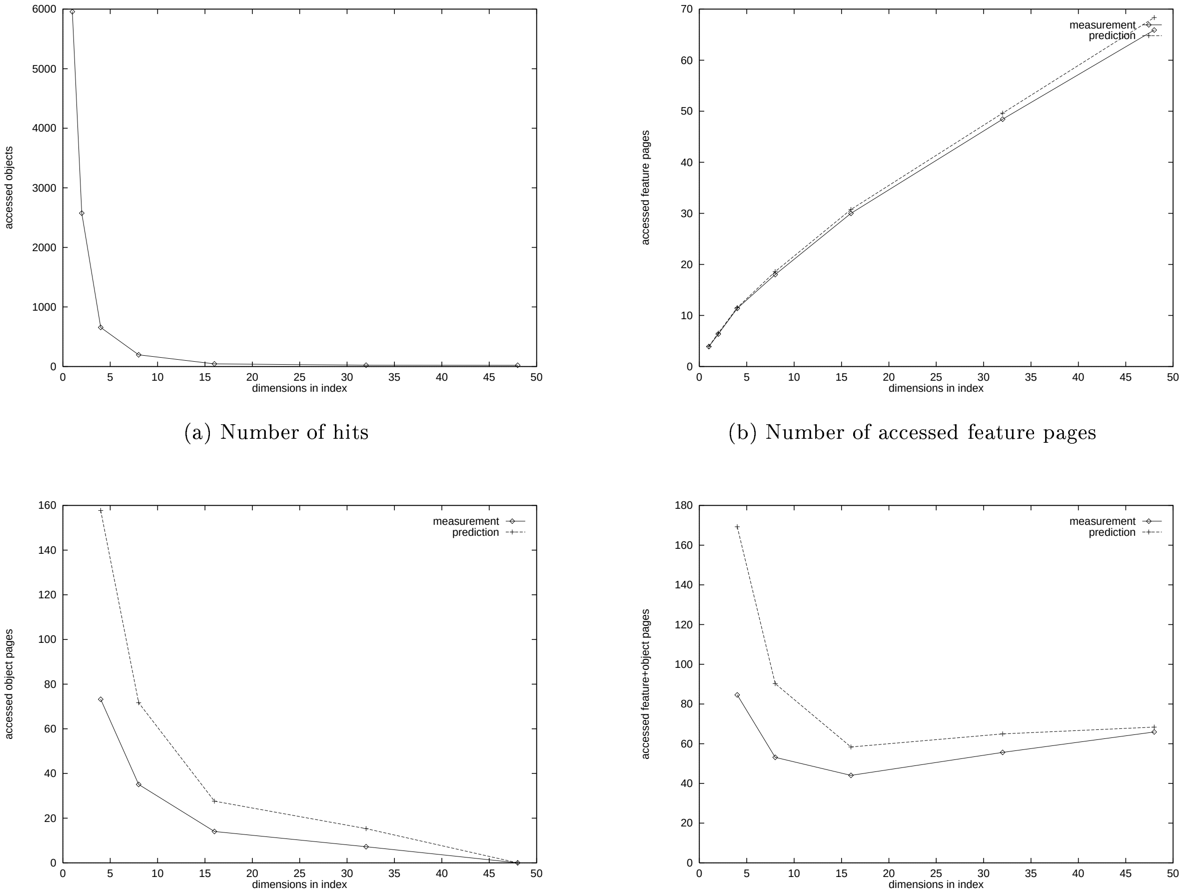
5.2.2 Computing the number of accessed object pages

The number of accessed object pages depends directly on the number of retrieved objects (*hits*) from the object server[3]. The problem we need to solve is:

Given h hits and p object pages, what is the average number of accessed object pages under the assumption that the hits are distributed uniformly among the object pages?

To get the average number of page accesses, we have to multiply each possible number of page accesses with its probability and the number of ways this subset of pages

[3]Since the hits is the number of objects that lie in the query sphere of the reduced dimensions, we can obtain this number with one additional scan over the dataset.



(a) Number of hits

(b) Number of accessed feature pages

(c) Number of accessed object pages

(d) Overall number of accessed pages

Figure 12: Measurement and prediction for TEXTURE48 (SVD-transformed)

can be picked:

$$\sum_{i=1}^{p} \binom{pages}{i} P(p,i,h)i \tag{8}$$

where $P(pages, i, h)$ denotes the probability to access exactly i of the p pages with h hits.

In order to establish a formula for $P(p,i,h)$, note that this value is the same as the number of possible non-empty partitionings of h into i partitions divided by the total number of possible non-empty partitionings of h. The number of non-empty partitionings of h into i partitions is known as the *Stirling number of the second kind* $S(h,i)$ and is defined as

$$S(h,i) = iS(h-1,i) + S(h-1,i-1)$$
$$\text{with } S(n,0) = 0 \text{ for } n \geq 1 \text{ and } S(n,n) = 1 \text{ for } n \geq 0 \tag{9}$$

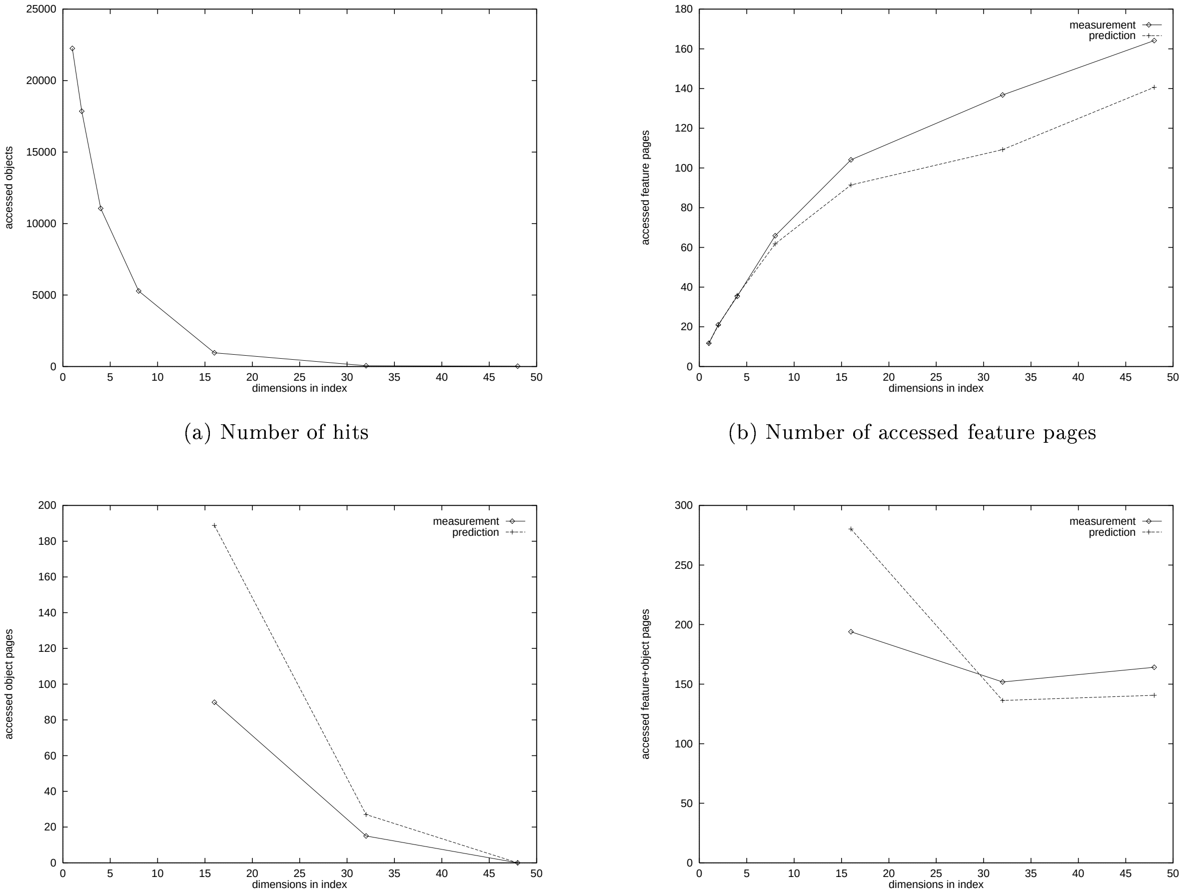
Therefore, $P(p,i,h)$ can be written as

$$P(p,i,h) = \begin{cases} \frac{S(h,i)}{\sum_{j=1}^{pages} \binom{p}{j} S(h,j)} & \text{if } h \geq i \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

By substituting $P(p,i,h)$ in Equation 8, we obtain a formula to compute the average number of accessed object pages.

5.3 Experimental Results

In what follows, we compare the predicted and measured costs (in number of disk pages) as the feature space dimensionality is varied. The measurements were done by building a VAMSplit R-tree [WJ96a] index on disk and running 500 21-NN queries on it. For the benefit of observing how good the predictions for feature pages and object pages are, we separate them into two different graphs. In addition, we also plot the number of hits as the feature space



(a) Number of hits

(b) Number of accessed feature pages

(c) Number of accessed object pages

(d) Overall number of accessed pages

Figure 13: Measurement and prediction for TEXTURE48 (non-SVD-transformed)

dimensionality is varied. In summary, for each dataset we present four graphs: the first one shows the number of accessed objects or hits, the second and third compare our predictions with the experimentally measured number of feature and object pages, respectively. The fourth graph compares the overall predicted cost with the overall measured cost.

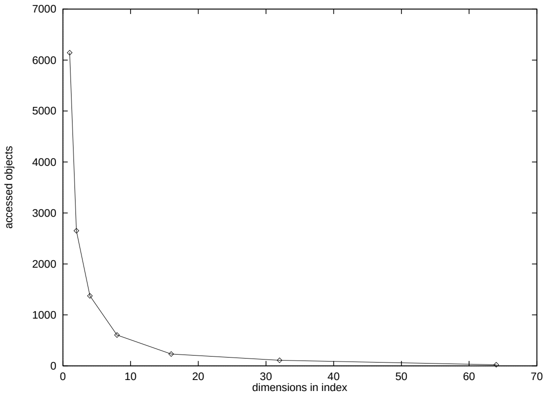
TEXTURE48 First, we examine the results for the (SVD-transformed) TEXTURE48 dataset. Figures 12(a) to 12(d) plot these results as a function of the feature space dimensionality. Since the data is SVD-transformed, most information is concentrated in the first 10 dimensions. As a result, the number of hits levels off nearly around this point. This is observed in Figure 12(a).

The number of accessed feature pages in Figure 12(b) is a monotonically increasing function. The reason for this behavior is the decrease in page capacity for higher dimensions. So, even though less objects have to be inspected, the number of accessed feature pages still increases. Our model predicts a curve that differs only marginally from the measurement.

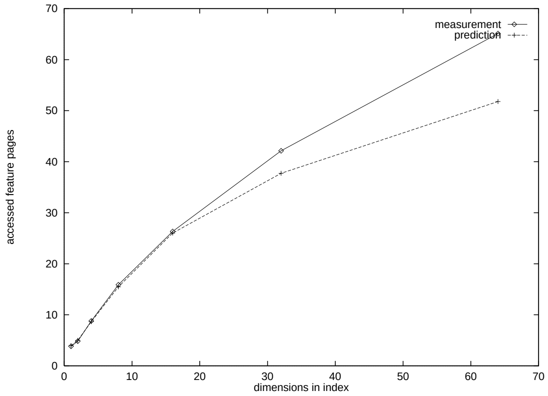
Figure 12(c) compares the predicted and the measured object page accesses. As expected, the number of object pages falls relatively fast, reflecting the behavior of the hits curve. Our model also predicts a fast drop, but the number of predicted accessed object pages for lower dimensions is too high. This is due to the fact that we assume uniformly distributed hits within the object pages. In reality, however, this assumption seems not to hold, leading to a discrepancy between experiment and prediction whenever the number of hits is high.

The overall cost in Figure 12(d) shows a slightly u-shaped curve with a minimum between 8 and 16 dimensions indicating that this is the right point to build the index. Our model predicts the same trend and has the same minimum.

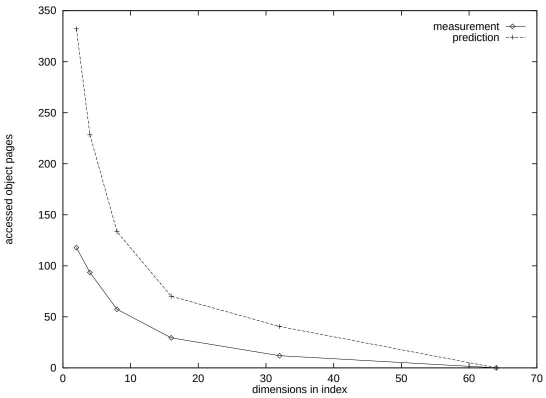
TEXTURE48 (not transformed) Next, we present the results for the non-transformed TEXTURE48 dataset.



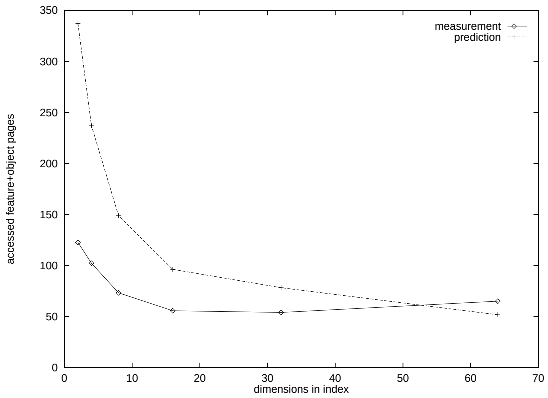
(a) Number of hits



(b) Number of accessed feature pages



(c) Number of accessed object pages



(d) Overall number of accessed pages

Figure 14: Measurement and prediction for subset of COLOR64 (SVD-transformed)

Unlike the SVD-transformed data where most variance is in the first 10 dimensions, the variance for the non-transformed data is spread over 30 dimensions. Consequently, the curve for the number of accessed objects drops at a much slower rate as shown in Figure 13(a).

As seen in Figure 13(b), the number of accessed feature pages increases rapidly till 15 dimensions and increases slower after this point. Note also that the number of accessed feature pages is much higher than for the previous dataset. This is due to a higher search radius. Our model predicts this trend very well. The prediction curve also bends at 15 dimensions and the values have the same magnitude. However, we predict slightly too slow values for high dimensions. This is caused by our uniformity assumption when adjusting the page sizes after sampling. Since the pages are grown by a too small value, fewer of them are intersected by the query spheres.

The curve for the accessed object pages shown in Figure 13(c) reflects the hits curve of Figure 13(a). Our model predicts the right trend, but the predicted values are once again too high for lower dimensions. This is again because of the assumption of uniformly distributed hits within the object pages. We present only three prediction values here since the calculation time of the Stirling numbers grows immensely for smaller dimensions. We plan on working on faster estimations in the future.

When looking at the overall cost in Figure 13(d), the curve is also u-shaped but has its minimum at 32 dimensions. Our model gives the same trend but predicts too high values at 16 dimensions. This is caused by the over-estimation of the object page accesses.

COLOR64 Next, we present the results for the SVD-transformed COLOR64 dataset. As seen in Figure 14(a), the number of hits drops fast through the first 20 dimensions. This is because most of the information is concentrated in those dimensions. After that, the number of hits levels off to the value of k, in this case 21.

As seen in Figure 14(b), the number of accessed feature pages is a monotonically increasing function. Our model predicts this trend initially, but for dimensionalities higher than 20, it again gives slightly too low values.

Figure 14(c) shows that the predicted object page accesses are again too high for low dimensions. As explained earlier, this is because of the assumption of uniformly distributed hits within the object pages.

The overall cost as seen from Figure 14(d) is a slightly u-shaped curve. The minimum cost lies between 15 and 35 dimensions. Our prediction shows the initial drop but not the upward trend towards the end. This is caused by the error in the feature page predictions. However, the gain in page accesses by choosing 25 rather than 64 index dimensions (as predicted) is only marginal in this example anyway.

6 Application 2: Determining Optimal Index Structure

As a second application of our sampling-based model, we show how it can be used to determine the optimal (in the sense of page accesses) index structure for a given dataset. In our experiment we chose the TEXTURE48 dataset and built two index structures for it: a VAMSplit R-tree and a k-d-tree. Then we measured the number of average page accesses for both structures by running 500 21-NN queries and compared it to the predictions.

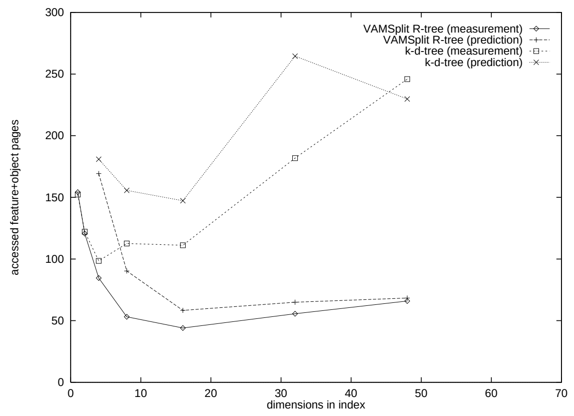


Figure 15: Measurement and prediction for different index structures (TEXTURE48)

Measurement and prediction show the same overall behavior (cf. Figure 15): for smaller dimensions, the page accesses are very similar but for higher dimensions the VAMSplit R-tree requires noticeably fewer page accesses (40 versus 100 for the k-d-tree). For this dataset, it is therefore better to choose a VAMSplit R-tree index structure in order to access the data. Our model predicted similar values (60 versus 150 page accesses) including the correct dimensionality to build the index at. However, instead of running experiments which may take several hours, we can reach this conclusion with our prediction within minutes.

7 Conclusions

In this paper, we presented two new models for predicting the performance of index structures for high-dimensional data. While other approaches restrict their applicability to low-dimensional or uniform datasets, our model works well for uniform as well as clustered non-uniform data. Due to the use of the original search radius, our algorithm automatically handles boundary effects that are otherwise hard to predict.

The comparison showed that the density-based prediction results in higher relative errors but has the advantage of being only marginally dependent of the index structure used. The sampling-based model delivers the best results but requires thorough knowledge about the underlying index.

Finally, we demonstrated with real dataset that it is possible to predict such complex non-linear behavior as the number of page accesses with high accuracy. These findings can be used to build index structures at the optimal number of reduced dimensions. A more detailed discussion with more examples can be found in [LS99].

We are planning on investigating the effect of different index structures (such as SS-trees or SR-trees) on our model. Another open question is whether this model works for other kinds of queries, such as range queries. Finally, we will experiment with general metric spaces.

References

[BBK98] S. Berchtold, C. Bohm, and H.-P. Kriegel. The pyramid technique: Towards breaking the curse of dimensionality. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 142–153, 1998.

[BBKK97] S. Berchtold, C. Bohm, D. A. Keim, and H.-P. Kriegel. A cost model for nearest neighbor search in high-dimensional data space. In *Proc. ACM Symp. on Principles of Database Systems*, 1997.

[Ben75] J. L. Bentley. Multi-dimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[BKK96] S. Berchtold, D. A. Keim, and H. P. Kriegel. The X-tree: An index structure for high dimensional data. *Proceedings of the Int. Conf. on Very Large Data Bases*, 1996.

[BKSS90] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 322–331, May 23-25 1990.

[CM99] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. Int. Conf. on Data Engineering*, pages 440–447, February 1999.

[FK94] Christos Faloutsos and Ibrahim Kamel. Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension. In *Proc. ACM Symp. on Principles of Database Systems*, pages 4–13, 1994.

[Fre87] M. W. Freeston. The BANG file: a new kind of grid file. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, pages 260–269, May 1987.

[Fre95] M. W. Freeston. A general solution of the n-dimensional B-tree problem. *Proc. of the ACM SIGMOD Intl. Conf. on Management of Data*, May 1995.

[Fre96] M. W. Freeston. A new generic index technology. *Proc. NASA Goddard Conf. on Mass Storage Technologies*, September 1996.

[Gut84] A. Guttman. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 47–57, 1984.

[KS97] N. Katayama and S. Satoh. The SR-tree: An index structure for high-dimensional nearest-neighbor queries. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, May 1997.

[KSF+96] F. Korn, N. Sidirupolous, C. Faloutsos, E. Seigel, and Z. Protopapas. Fast nearest neighbor search in medical image databases. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 215–226, 1996.

[LJF94] King-Ip Lin, H. V. Jagdish, and C. Faloutsos. The TV-tree: An index structure for high-dimensional data. *VLDB Journal*, 3:517–542, 1994.

[LS90] D. B. Lomet and B. Salzberg. The hB-tree: A multi-attribute indexing method with good guaranteed performance. *Proc. ACM Symp. on Transactions of Database Systems*, 15(4):625–658, December 1990.

[LS99] C. A. Lang and A. K. Singh. Predicting the performance of index structures for high-dimensional datasets. Technical Report CS-TR-9934, Univ. of California at Santa Barbara, 1999.

[Ozs97] Meral Ozsoyglu. The M-tree: An index structure for metric spaces. *Proceedings of the Int. Conf. on Very Large Data Bases*, 1997.

[PM97] Apostolos Papadopoulos and Yannis Manolopoulos. Performance of nearest neighbor queries in R-trees. In *ICDT*, pages 394–408, 1997.

[SK98] T. Seidl and H.-P. Kriegel. Optimal multi-step k-nearest neighbor search. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1998.

[STC97] S. Sclaroff, L. Taycher, and M. La Cascia. Imagerover: A content-based image browser for the world wide web. In *Proc. IEEE Workshop on Content-based Access of Image and Video Libraries*, June 1997.

[WBKW96] E. Wold, T. Blum, D. Keislar, and J. Wheaton. Content-based classification, search, and retrieval of audio. *IEEE Multimedia*, 3(3):27–36, Fall 1996.

[WJ96a] D. White and R. Jain. Algorithms and strategies for similarity retrieval. *Proc. of the SPIE Conference*, 1996.

[WJ96b] D. White and R. Jain. Similarity indexing with the SS-tree. *Proc. Int. Conf. on Data Engineering*, pages 516–523, 1996.

[WSB98] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the Int. Conf. on Very Large Data Bases*, August 1998.