Proc. IEEE Int. Conf. on Database Theory (ICDT), 2001

On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces

Stefan Berchtold[1], Christian Böhm[2], Daniel Keim[3], Florian Krebs[2], Hans-Peter Kriegel[2]

[1] stb gmbh, Ulrichsplatz 6, 86150 Augsburg, Germany
{Stefan.Berchtold}@stb-gmbh.de
[2] University of Munich, Oettingenstr. 67, 80538 Munich, Germany
{boehm,kriegel}@informatik.uni-muenchen.de
[3] University of Halle-Wittenberg, Kurt-Mothes Str. 1, 06099 Halle (Saale), Germany
keim@informatik.uni-halle.de

Abstract. Nearest-neighbor queries in high-dimensional space are of high importance in various applications, especially in content-based indexing of multimedia data. For an optimization of the query processing, accurate models for estimating the query processing costs are needed. In this paper, we propose a new cost model for nearest neighbor queries in high-dimensional space, which we apply to enhance the performance of high-dimensional index structures. The model is based on new insights into effects occurring in high-dimensional space and provides a closed formula for the processing costs of nearest neighbor queries depending on the dimensionality, the block size and the database size. From the wide range of possible applications of our model, we select two interesting samples: First, we use the model to prove the known linear complexity of the nearest neighbor search problem in high-dimensional space, and second, we provide a technique for optimizing the block size. For data of medium dimensionality, the optimized block size allows significant speed-ups of the query processing time when compared to traditional block sizes and to the linear scan.

1. Introduction

Nearest neighbor queries are important for various applications such as content-based indexing in multimedia systems [10], similarity search in CAD systems [7, 17], docking of molecules in molecular biology [21], and string matching in text retrieval [1]. Most applications use some kind of feature vector for an efficient access to the complex original data. Examples of feature vectors are color histograms [20], shape descriptors [15, 18], Fourier vectors [23], and text descriptors [14]. According to [3], nearest neighbor search on high-dimensional feature vectors may be defined as follows:

Given a data set *DS* of points in a *d*-dimensional space $[0, 1]^d$, find the data point *NN* from *DS* which is closer to the given query point *Q* than any other point in the *DS*. More formally:

$$NN(Q) = \{\bar{e} \in DS \mid \forall e \in DS: \|\bar{e} - Q\| \leq \|e - Q\|\}.$$

A problem of index-based nearest neighbor search is that it is difficult to estimate the time which is needed for executing the nearest neighbor query. The estimation of the time, however, is crucial for optimizing important parameters of the index structures such as the block size. An adequate cost model should work for data sets with an arbitrary number of dimensions and an arbitrary size of the database, and should be applicable to different data distributions and index structures. Most important, however, it should provide accurate estimates of the expected query execution time in order to allow an optimization of the parameters of the index structure.

In a previous paper [3], we proposed a cost model which is very accurate for estimating the cost of nearest-neighbor queries in high-dimensional space. This cost model is based on ta-

bles which are generated using the Montecarlo integration method. Although expensive numerical integration occurs only in the compile time of the cost estimator, not in the execution time (when actually determining the cost of query execution), expensive numerical steps are completely avoided in the cost model proposed in this paper. Based on recent progress in understanding the effects of indexing high-dimensional spaces [4], we develop a new cost model for an index-based processing of nearest neighbor queries in high-dimensional space. The model is completely analytical allowing a direct application to query optimization problems. The basic idea of the model is to estimate the number of data pages intersecting the nearest neighbor sphere by first determining the expected radius of the sphere. Assuming a certain location of the query point and a partition of the data space into hyperrectangles, the number of intersected pages can be represented by a staircase function. The staircase function results from the fact that the data pages can be collected into groups such that all pages in a group have the same "skewness" to the query point. Each page in the group is intersected simultaneously and, therefore, the cost model results in a staircase function.

The model has a wide range of interesting theoretical and practical applications. The model may, for example, be used to confirm the known theoretical result [24] that the time complexity of nearest neighbor search in a very high dimensional space is linear. Since the model provides a closed formula for the time complexity (depending on the parameters: dimensionality, database size, and block size), the model can also be used to determine the practically relevant break-even dimensionality between index-based search and linear scan (for a given database size). For dimensionalities below the break-even point, the index-based search performs better, for dimensionalities above the break-even point, the linear scan performs better. Since the linear scan of the database can also be considered as an index structure with an infinite block size, the query optimization problem can also be modeled as a continuous block size optimization problem. Since our model can be evaluated analytically and since the block size is one of the parameters of the model, the cost model can be applied to determine the block size for which the minimum estimated costs occur. The result of our theoretical analysis is surprising and shows that even for medium-dimensional spaces, a large block size (such as 16kB for 16 dimensions) clearly outperforms traditional block sizes (2 kByte or 4 KByte) and the linear scan. An index structure built with the optimal block size always shows a significantly better performance, resulting in traditional block sizes in lower dimensional spaces and to a linear scan in very high dimensions. Also, the query processing cost will never exceed the cost for a linear scan, as index structures typically do in very high dimensions due to a large number of random seek operations. A practical evaluation and comparison to real measurements confirms our theoretical results and shows speed-ups of up to 528% over the index-based search and up to 500% over the linear scan. Note that a cost model such as the one proposed in [3] can hardly be used for a parameter optimization because of the numerical component of the model.

2. Related Work

The research in the field of nearest neighbor search in high-dimensional space may be divided into three areas: index structures, nearest neighbor algorithms on top of index structures, and cost models.

The first index structure focusing on high-dimensional spaces was the TV-tree proposed by Lin, Jagadish, and Faloutsos [16]. The basic idea of the TV-tree is to divide attributes into attributes which are important for the search process and others which can be ignored because these attributes have a small chance to contribute to query processing. The major drawback of the TV-tree is that information about the behavior of single attributes, e.g. their selectivity, is required. Another R-tree-like high-dimensional index structure is the SS-tree [25]

which uses spheres instead of bounding boxes in the directory. Although the SS-tree clearly outperforms the R*-tree, spheres tend to overlap in high-dimensional spaces and therefore, the performance also degenerates. In [13], an improvement of the SS-tree has been proposed, where the concepts of the R-tree and SS-tree are integrated into a new index structure, the SR-tree. The directory of the SR-tree consists of spheres (SS-tree) and hyperrectangles (R-tree) such that the area corresponding to a directory entry is the intersection between the sphere and the hyperrectangle. Another approach has been proposed in [6]. The X-tree is an index structure adapting the algorithms of R*-trees to high-dimensional data using two techniques: First, the X-tree introduces an overlap-free split algorithm which is based on the split history of the tree. Second, if the overlap-free split algorithm would lead to an unbalanced directory, the X-tree omits the split and the corresponding directory node becomes a so-called supernode. Supernodes are directory nodes which are enlarged by a multiple of the block size. Another approach related to nearest neighbor query processing in high-dimensional space is the parallel method described in [4]. The basic idea of the declustering technique is to assign the buckets corresponding to different quadrants of the data space to different disks, thereby allowing an optimal speed-up for the parallel processing of nearest neighbor queries.

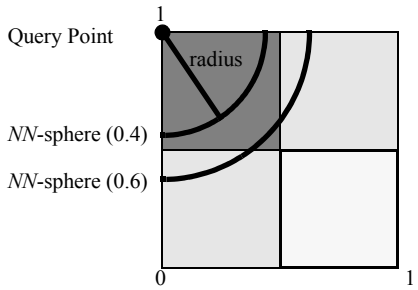


Figure 1: Two-dimensional Example for the Data Pages Affected by the Nearest Neighbor Search for an Increasing *NN*-distance

Besides the index structures, the algorithms used to perform the nearest neighbor search are obviously important. The algorithm of Rousopoulos et. al. [19] operates on R-trees. It traverses the tree in a top-down fashion, always visiting the closest bounding box first. Since in case of bounding boxes there always exists a maximal distance to the closest point in the box, the algorithm can prune some of the branches early in the search process. The algorithm, however, can be shown to be suboptimal since, in general, it visits more nodes than necessary, i.e., more nodes than intersected by the nearest neighbor sphere. An algorithm which avoids this problem is the algorithm by Hjaltason and Samet [12]. This algorithm traverses the space partitions ordered by the so-called MINDIST which is the distance of the closest point in the box to the query point. Since the algorithm does not work in a strict top-down fashion, the algorithm has to keep a list of visited nodes in main memory. The algorithm can be shown to be time-optimal [3]; however, in high-dimensional spaces, the size of the list, and therefore the required main memory, may become prohibitively large.

The third related area are cost models for index-based nearest neighbor queries. One of the early models is the model by Friedman, Bentley, and Finkel [11]. The assumptions of the model, however, are unrealistic for the high-dimensional case, since N is assumed to converge to infinity and boundary effects are not considered. The model by Cleary [9] extends the Friedman, Bentley, and Finkel model by allowing non-rectangular-bounded pages, but still does not account for boundary effects. Sproull [22] uses the existing models for optimizing the nearest neighbor search in high dimensions and shows that the number of data points must be exponential in the number of dimensions for the models to provide accurate estimates. A cost model for metric spaces has recently been proposed in [8]. The model, however, has been designed for a specific index-structure (the M-tree) and only applies to metric

spaces. In [3], a cost model has been proposed which is very accurate even in high-dimensions as the model takes boundary effects into account. The model is based on the concept of the Minkowsky-sum which is the volume created by enlarging the bounding box by the query sphere. Using the Minkowsky-sum, the modeling of the nearest neighbor search is transformed into an equivalent problem of modeling point queries. Unfortunately, when boundary effects are considered, the Minkowsky-sum can only be determined numerically by Montecarlo integration. Thus, none of the models can be used to optimize the parameters of high-dimensional indexing techniques in a query processor.

3. A Model for the Performance of Nearest Neighbor Queries in High-Dimensional Space

As a first step of our model, we determine the expected distance of the query point to the actual nearest neighbor in the database. For simplification, in the first approximation we assume uniformly distributed data[1] in a normalized data space $[0,1]^d$ having a volume of 1. The nearest neighbor distance may then be approximated by the volume of the sphere which on the average contains one data point. Thus,

$$Vol_{Sp}^{d}(NN\text{-}dist) = \frac{1}{N} \quad \Leftrightarrow \quad \frac{\sqrt{\pi}^{d}}{\Gamma(d/2+1)} \cdot NN\text{-}dist^{d} = \frac{1}{N}$$

where $\Gamma(n)$ is the gamma function ($\Gamma(x+1) = x \cdot \Gamma(x)$, $\Gamma(1) = 1$ and $\Gamma(1/2) = \sqrt{\pi}$), which may be approximated by $\Gamma(n) \approx (n/e)^{n} \cdot \sqrt{2 \cdot \pi \cdot n}$. From the above equation, the expected nearest neighbor distance may be determined as

$$NN\text{-}dist(N, d) = \sqrt[d]{\frac{\Gamma(d/2+1)}{N \cdot \sqrt{\pi}^{d}}} = \frac{1}{\sqrt{\pi}} \cdot \sqrt[d]{\frac{\Gamma(d/2+1)}{N}}$$

In general, the number of data points is not growing exponentially, which means that not all dimensions are used as split axes. Without loss of generality, we assume that the first d' dimensions have been used as split dimensions. Thus, d' may be determined as

$$d' = \left\lceil \log_2\left(\frac{N}{C_{eff}}\right) \right\rceil .$$

For simplification, we further assume that each of the d' split dimensions has been split in the middle.

To determine the number of pages which are intersected by the nearest neighbor sphere, we now have to determine the number of pages depending on *NN-dist(N,d)*. In our first approximation, we only consider the simple case that the query point is located in a corner of the data space[2]. In figure 1, we show a two-dimensional example for the data pages which are affected by the nearest neighbor search for an increasing *NN-dist(N, d)*. Since the data space is assumed to be normalized to $[0,1]^{d}$ and since the data pages are split at most once, we have to consider more than one data page if $NN\text{-}dist(N, d) \geq 0.5$. In this case, two additional data pages have to be accessed in our two-dimensional example (cf. figure 1). If *NN-dist(N, d)* increases to more than $0.5 \cdot \sqrt{2}$, we have to consider even more data pages, namely all four

1. In section 5, we provide an extension of the model for non-uniform data distributions.
2. In high-dimensional space, this assumption is not unrealistic since most of the data points are close to the surface of the data space (for details see [5]).

pages in our example. In the general case, we have to consider more data pages each time the *NN-dist(N,d)* exceeds the value $0.5 \cdot \sqrt{i}$. We therefore obtain

$$NN\text{-}dist(N, d) = 0.5 \cdot \sqrt{i} \qquad \text{(for } i = 1 \ldots d\text{')}$$

$$\Leftrightarrow \quad i = \left(\frac{NN\text{-}dist(N, d)}{0.5} \right)^2 \qquad \text{(for } i = 1 \ldots d\text{')}$$

$$\Leftrightarrow \quad i = \left(\frac{\frac{1}{\sqrt{\pi}} \cdot \sqrt[d]{\frac{\Gamma(d/2 + 1)}{N}}}{0.5} \right)^2 \qquad \text{(for } i = 1 \ldots d\text{')}$$

$$\Rightarrow \quad i \approx \frac{2 \cdot (d + 2)}{e \cdot \pi} \cdot \sqrt[d]{\frac{\pi \cdot (d + 2)^3}{4 \cdot e^2 \cdot N^2}} \qquad \text{(for } i = 1 \ldots d\text{')}$$

The number of data pages which have to be considered in each step are the data pages that differ in i of the d' split dimensions, which may be determined as $\binom{d'}{i}$.

d	dimension
N	number of data points
$\|db\|$	size of the database
$\#b$	number of data pages in the database
$\|b\|$	page size
u	storage utilization
d'	number of split dimensions
C_{eff}	average number of data points per index page
T_{IO}	I/O time (disc access time independent from the size of the accessed data block)
T_{Tr}	transfer and processing time (linearly depends on the size of the accessed data block)
NN-dist($\|db\|$, d)	nearest neighbor distance depending on $\|db\|$ and d
P(N,d)	number of data pages depending on N and d
P(|db|, d*)*	number of data pages depending on $\|db\|$ and d
X(#b)	percentage by which the linear scan is faster

Table 1: Important Symbols

Integrating the formulas provides the number of data pages which have to accessed in performing a nearest neighbor query on a database with N data points in a d-dimensional space:

$$P(N, d) = \sum_{k=0}^{\frac{2\cdot(d+2)}{e\cdot\pi}\cdot\sqrt[d]{\frac{\pi\cdot(d+2)^3}{4\cdot e^2\cdot N^2}}} \binom{d'}{k} .$$

In the following, we determine the development of $P(N, d)$ for a constant size database ($|db| = const$). In this case, the number of data pages is also constant

$$\#b = \frac{|db|}{|b|\cdot u}$$

and the number of data points linearly depends on the database size $|db|$ and the dimensionality d

$$N = \frac{|db|}{d} .$$

The nearest neighbor distance *NN-dist* now depends on the dimensionality (d) and the size of the database ($|db|$)

$$NN\text{-}dist(|db|, d) = \frac{1}{\sqrt{\pi}}\cdot\sqrt[d]{d\cdot\Gamma(d/2+1)\cdot|db|}$$

and since $C_{eff} = |b|\cdot u/d$, the number of split dimensions d' becomes

$$d' = \left\lceil \log_2\left(\frac{N}{C_{eff}}\right) \right\rceil = \left\lceil \log_2\left(\frac{|db|}{|b|\cdot u}\right) \right\rceil .$$

The number of data pages which have to be accessed in performing a nearest neighbor query in d-dimensional space can now be determined as

$$P(|db|, d) = \sum_{k=0}^{\frac{2\cdot(d+2)}{e\cdot\pi}\cdot\sqrt[d]{\frac{\pi\cdot d^2\cdot(d+2)^3}{4\cdot e^2\cdot|db|^2}}} \binom{\left\lceil \log_2\left(\frac{|db|}{|b|\cdot u}\right) \right\rceil}{k}$$

In figure 2, we show the development of $P(|db|, d)$ depending on the dimensionality d. $P(|db|, d)$ is a staircase function which increases to the maximum number of data pages ($\#b$). The staircase property results from the discontinuous increase of $P(|db|, d)$ each time

$$NN\text{-}dist(N, d) \geq 0.5\cdot\sqrt{i} \quad (for \;\; i = 1 \ldots d') ,$$

which is a consequence of using the corners of the data space as query points. Since for practical purposes this assumption is not sufficient, in section 5 we extend the model to consider other query points as well.

4. Evaluation

After introducing our analytical cost model, we are now able to compare the time needed to perform an index-based nearest neighbor search with the sequential scan of the database. Let us first consider the factor by which a sequential read is faster than a random block-wise access. Given a fixed size database ($|db|$) consisting of $\#b = |db|/(|b|\cdot u)$ blocks, the time needed to read the database sequentially is $T_{IO} + \#b\cdot T_{Tr}$. Reading the same database in a

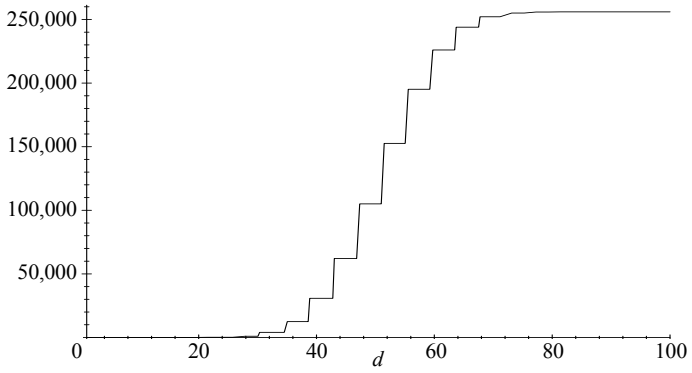


Figure 2: Number of Page Accesses as Determined by our Cost Model

block-wise fashion requires $\#b \cdot (T_{IO} + T_{Tr})$. From that we get the factor X, by which the sequential read is faster, as

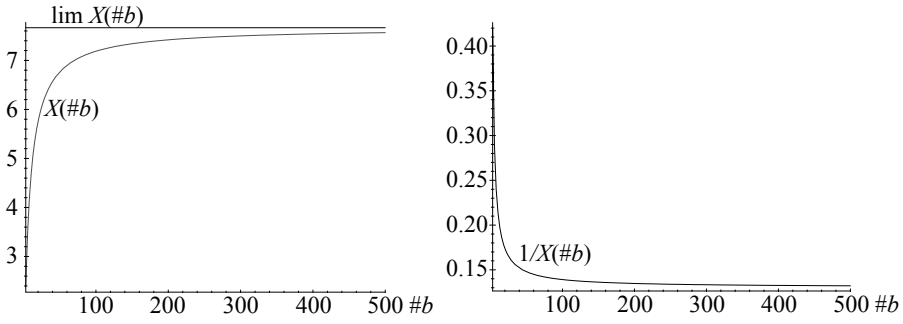
$$X(\#b) \cdot \#b \cdot (T_{IO} + T_{Tr}) = T_{IO} + \#b \cdot T_{Tr} \quad \Rightarrow \quad X(\#b) = \frac{\#b \cdot (T_{IO} + T_{Tr})}{T_{IO} + \#b \cdot T_{Tr}}$$

In figure 3a, we show the development of *X(#b)* depending on the size of the database for realistic system parameters which we measured in our experiments (T_{I0} = 10 ms, T_{Tr} = 1.5 ms). Note that for very large databases, *X(#b)* converges against a system constant

$$\lim_{\#b \to \infty} X(\#b) = \frac{T_{IO} + T_{Tr}}{T_{Tr}} .$$

It is also interesting to consider the inverse of *X(#b),* which is the percentage of blocks that could be read randomly instead of sequentially reading the whole database. The function $1/(X(\#b))$ is plotted in figure 3b for a constant database size.

Let us now compare the time needed to perform an index-based nearest neighbor search with the time needed to sequentially search the database. The time needed to read the database



a. Factor by which seq. scan is faster

b. Percentage of pages which can be read randomly instead of seq. scan

Figure 3: Development of *X(#b)* and *1/X(#b)*

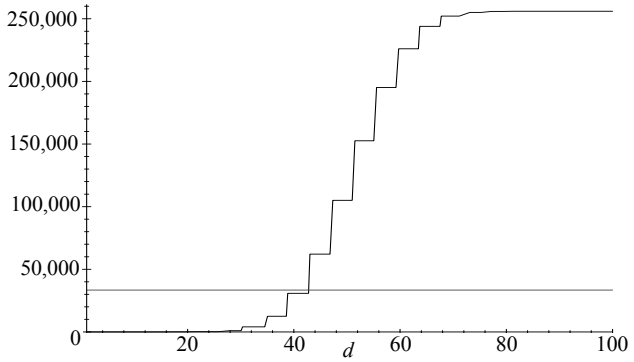


Figure 4: Comparison of Index-based Search and the Sequential Scan Depending on the Dimensionality *d* for a Fixed-Size Database

sequentially is $T_{IO} + \#b \cdot (T_{Tr} + T_{CPU})$, and the time needed to randomly access the necessary pages in an index-based search is

$$T_{IndexSearch}(|db|, d) = (T_{IO} + T_{Tr}) \cdot \sum_{k=0}^{\frac{2\cdot(d+2)}{e\cdot\pi}\cdot\sqrt[d]{\frac{\pi\cdot d^2\cdot(d+2)^3}{4\cdot e^2\cdot|db|^2}}} \left(\left\lceil \log_2\left(\frac{|db|}{|b|\cdot u}\right) \right\rceil \right)$$

In figure 4, the time resulting from the two formulas is shown for a fixed database size of 256,000 blocks. Note that in the example for $d = 44$, the linear scan becomes faster than the index-based search. In figure 5, we show the time development depending on both, the dimensionality and the database size. It is clear that for all realistic parameter settings, there exists a dimensionality d for which the linear scan becomes faster than the index-based search.

This fact is summarized in the following lemma:

Lemma: *(Complexity of Index-based Nearest Neighbor Search)*
For realistic system parameters (i.e., $T_{IO} > 0$, $|db| > |b|$, and $u > 0$), there always exists a dimension d , for which the sequential scan is faster than an index-based search. More formally,

$$\forall |db| \;\; \exists \tilde{d}: \;\; T_{IndexSearch}(|db|, \tilde{d}) \;\; > \;\; T_{LinScan}(|db|) .$$

Idea of the Proof:
From our previous observations, it is clear that $T_{IndexSearch}(|db|, \tilde{d})$ increases faster than $T_{LinScan}(|db|)$ and finally all data blocks of the database are read. This is the case for

$$\frac{2\cdot(\tilde{d}+2)}{e\cdot\pi}\cdot\sqrt[d]{\frac{\pi\cdot\tilde{d}^2\cdot(\tilde{d}+2)^3}{4\cdot e^2\cdot|db|^2}} \;=\; \log_2\left(\frac{|db|}{|b|\cdot u}\right),$$

in which case the index-based search randomly accesses all blocks of the database. A sequential read of the whole database is faster than the block-wise access by a factor of

$$X(\#b) \;=\; \frac{|db|\cdot(T_{IO}+T_{Tr})}{|b|\cdot u\cdot T_{IO} + |db|\cdot T_{Tr}}$$

and therefore, the correctness of the lemma is shown. q.e.d.

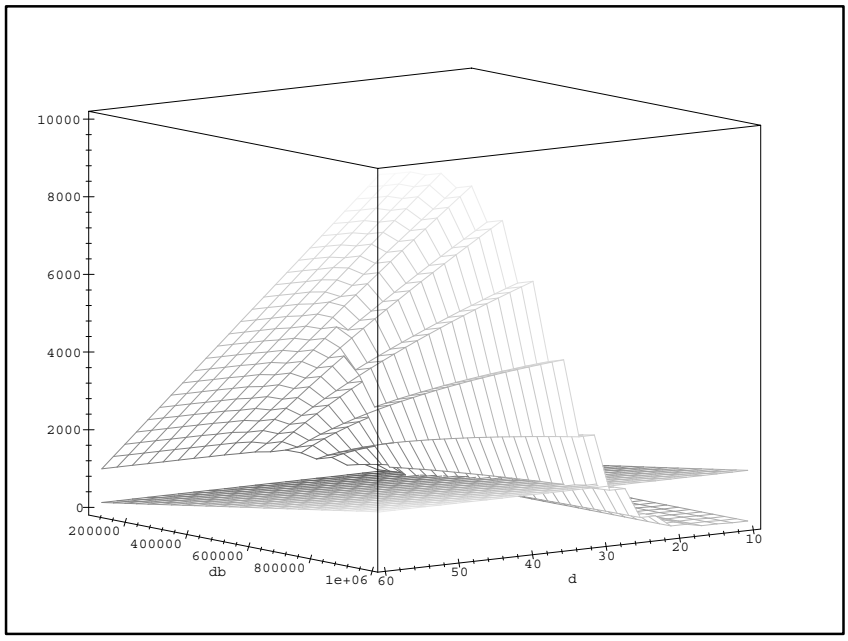


Figure 5: Comparison of Index-based Search and the Sequential Scan Depending on the Database Size |*db*| and the Dimensionality *d*

Note that this result confirms the pessimistic result by Weber et al. [24] that nearest neighbor search in high-dimensional space has a linear time complexity. In figure 6, we compare the performance estimations of our numerical model [3] and the new analytical model (proposed in this paper) to the performance of the R*-tree [2] on a uniformly distributed data set with a fixed number of data pages (256) and varying dimensionality (d = 2 ... 50). As one can see, the analytical cost model provides an accurate prediction of the R*-tree's performance over a wide range of dimensions. Even for low and medium dimensions, the prediction of our model is pretty close to the measured performance.

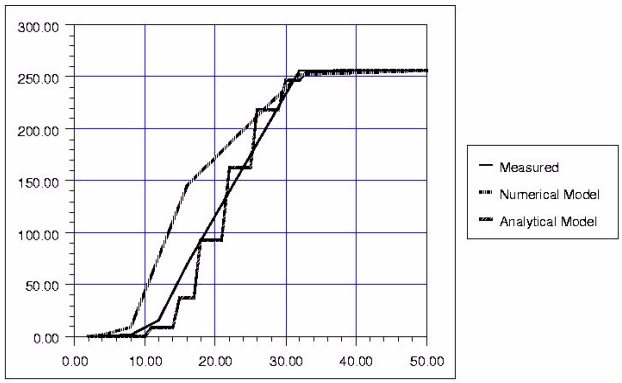


Figure 6: Comparison of Cost Models and Measured R*-Tree Performance Depending on the Dimension

5. Optimized Processing of Nearest Neighbor Queries in High-Dimensional Space

The objective of this section is to show, how the results obtained above can be used for an optimization of the query processing. We have shown in section 4 that data spaces exist, where it is more efficient to avoid an index-based search. Instead, the sequential scan yields a better performance. In contrast, for low-dimensional data spaces, multidimensional index structures yield a complexity which is logarithmic in the number of data objects. In this section, we show that it is optimal for data spaces with moderate dimensionality to use a logical block size which is a multiple of the physical block size provided by the operating system. Therefore, we have to consider three different cases of query processing: (1) Index-based query processing with traditional block size (4 KByte) in low-dimensional data spaces, (2) Sequential Scan processing in high-dimensional data spaces and (3) Index-based query processing with enlarged block-size in medium-dimensional data spaces. As the sequential scan can be considered as an index with infinite block size, the third case subsumes case (2). Case (1) is trivially subsumed by case (3). Therefore, the problem is reduced to a single optimization task of minimizing the access costs by varying the logical block size.

In order to obtain an accurate estimation of the minimum-cost blocksize, especially in the presence of non-uniform data distributions, we have to slightly extend our model. Hence, we express the expected location of the query point more accurately than assuming the query point to be in a corner of the data space. If we assume that the location of the query point is uniformly distributed, we are able to determine the expected distance *Edist* of the query point to the closest corner of the data space. However, the formula for *Edist* is rather complex and therefore not applicable for practical purposes. Instead, we use the following empirically derived approximation of the formula which is very accurate up to dimension 100:

$$Edist = \frac{d^{0.53}}{4}$$

Figure 7 compares the exact and the approximated distances demonstrating the good accuracy of the approximation.

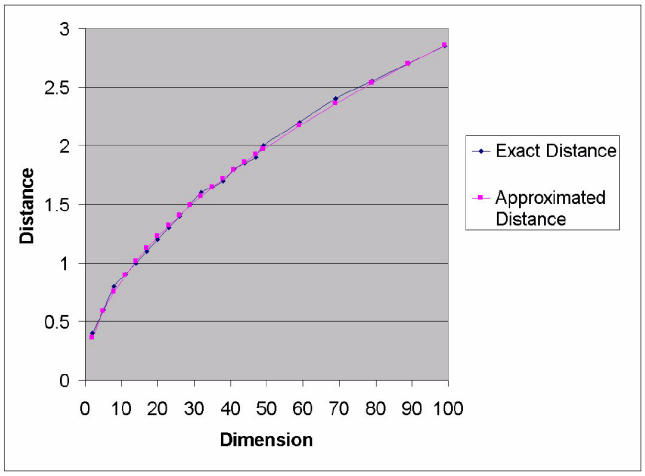


Figure 7: Approximated distance of the query point to the closest corner of data space

If we assume that the query point is located on the diagonal of the data space, we can adapt our model using the approximate value of *Edist*. In this case, we have to consider more data pages each time *NN-dist*(*N*, *d*) exceeds the value $\left(0.5 - \frac{d^{0.53}}{4\sqrt{d}}\right) \cdot \sqrt{i}$ rather than exceeding $0.5 \cdot \sqrt{i}$ in the original model. Thus, our extended cost function turns out as:

$$\frac{d+2}{e \cdot \pi \cdot \left(0.5 - \frac{d^{0.53}}{4\sqrt{d}}\right)^2} \cdot \sqrt[d]{\frac{\pi \cdot (d+2)^3 \cdot d^2}{4 \cdot |db|^2 \cdot e^2}} \cdot \left(\left\lceil \log_2\left(\frac{|db|}{|b| \cdot u}\right) \right\rceil \right)_k .$$

$k = 0$

Figure 8 depicts the graph of the extended cost model with varying dimension and varying block size. As the model is discrete over the dimensions, there are some staircases. On the other hand, the model is continuous over the blocksize and therefore, amenable to an optimi-

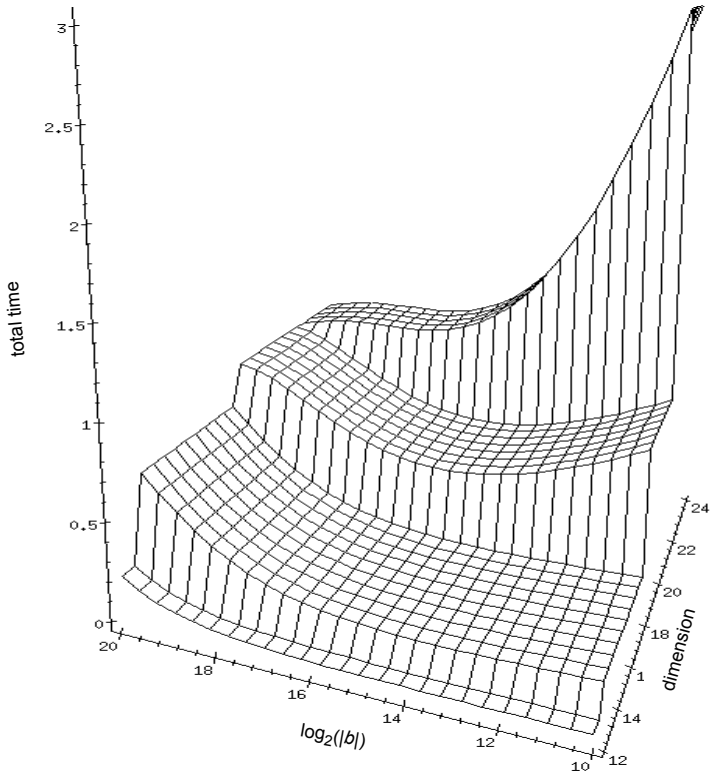


Figure 8: Graph of the Extended Cost Function: The staircase in the front is associated to dimensions for which a minimum block size yields least cost. The staircase in the back shows dimensions for which sequential scan causes less costs than index search whereas the staircase in the middle shows minimum cost for an optimal block size in a medium size range.

zation by differentiation. In Figure 8, all three cases can be seen: For low dimensions (staircase in the front), the cost function is monotonously increasing with increasing block size. Therefore, the lowest possible blocksize should be taken. In the high-dimensional case (staircase in the back), the cost function is monotonously decreasing with increasing block size. Therefore, the optimal blocksize is infinite, or in other words, we are supposed to use sequential scan instead of indexing.
The most interesting phenomenon is the staircase in the middle. Here the cost function is monotonously falling to a minimum and then monotonously increasing. The cost function has obviously a single minimum and no further local extrema, which facilitates the search for the optimum. In order to find the block size for which the minimum cost occur, we simply derive the cost function with respect to the block size. The derivative of the cost function $T_{\text{IndexSearch}}(|db|, d, |b|)$ can be determined, as follows:

$$\frac{\partial T_{\text{IndexSearch}}(|db|, d, |b|)}{\partial |b|} =$$

$$= \frac{\partial}{\partial |b|}(T_{\text{IO}} + |b| \cdot T_{Tr}) \cdot \sum_{k=0}^{\frac{d+2}{e \cdot \pi \cdot \left(0.5 - \frac{d^{0.53}}{4\sqrt{d}}\right)^2} \cdot \sqrt[d]{\frac{\pi \cdot (d+2)^3 \cdot d^2}{4 \cdot |db|^2 \cdot e^2}}} \left(\left\lceil \log_2\left(\frac{|db|}{|b| \cdot u}\right) \right\rceil \right)$$

$$= \sum_{k=0}^{\frac{d+2}{e \cdot \pi \cdot \left(0.5 - \frac{d^{0.53}}{4\sqrt{d}}\right)^2} \cdot \sqrt[d]{\frac{\pi \cdot (d+2)^3 \cdot d^2}{4 \cdot |db|^2 \cdot e^2}}} \frac{\partial}{\partial |b|} \left((T_{\text{IO}} + |b| \cdot T_{Tr}) \left(\left\lceil \log_2\left(\frac{|db|}{|b| \cdot u}\right) \right\rceil \right) \right)$$

$$= \sum_{k=0}^{\frac{d+2}{e \cdot \pi \cdot \left(0.5 - \frac{d^{0.53}}{4\sqrt{d}}\right)^2} \cdot \sqrt[d]{\frac{\pi \cdot (d+2)^3 \cdot d^2}{4 \cdot |db|^2 \cdot e^2}}} \left(\log_2\left(\frac{|db|}{|b| \cdot u}\right) \right) \cdot (T_{\text{Tr}} + (T_{\text{IO}} + |b| \cdot T_{\text{Tr}}) \cdot \%)$$

with $\% = \dfrac{\Psi\left(\log_2\left(\frac{|db|}{|b| \cdot u}\right) - k + 1\right) - \Psi\left(\log_2\left(\frac{|db|}{|b| \cdot u}\right) + 1\right)}{|b| \cdot \ln(2)}$, where Ψ is the well-known digamma function, the derivative of the natural logarithm of the Γ-function:

$$\Psi(x) = \frac{\partial}{\partial x} \ln(\Gamma(x)) .$$

The derivative of $T_{\text{IndexSearch}}$ is continuous over $|b|$. Three cases can be distinguished: (1) The derivative is positive over all block sizes. In this case, the minimum block size is optimal. (2) The derivative is negative. Then, an infinite blocksize is optimal and the search is processed by sequentially scanning the only data page. (3) The derivative of the cost function has a zero value. In this case, the equation

$$\frac{\partial T_{\text{IndexSearch}}(|db|, d, |b|)}{\partial |b|} = 0$$

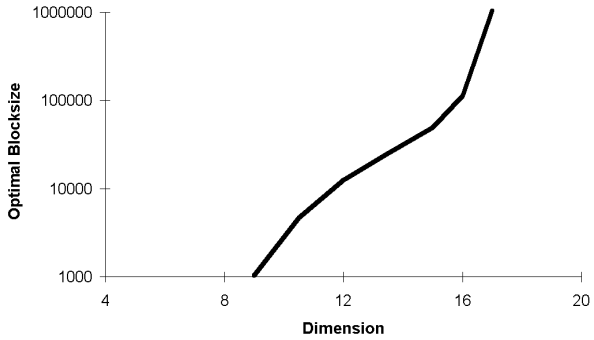


Figure 9: Optimal Block Size for Varying Dimensions

has to be solved, determining the optimal blocksize $|b| = |b|_{opt}$. Finding the optimal blocksize can be done by a simple binary search. As the cost function is smooth, this will lead to an optimal blocksize after a small (logarithmic) number of steps.

The development of the optimal block size over varying dimensions and varying database sizes is depicted in Figure 9. The position of the optimum is, as expected, heavily affected by the dimension of the data space. At dimensions below 9, it is optimal to use the minimal block size of the system, in this case 1KByte. In contrast, when searching in a 17-dimensional data space (or higher), the optimal block size is infinite. Therefore, the sequential scan yields the best performance in this case. The position of the optimum also depends on the database size. Figure 10 shows the optimal block size with varying the database size. Although the variation is not very strong, the optimal block size is decreasing with increasing the database size. We therefore suggest a dynamical adaption of the block size if the database size is unknown a priori.

Finally, we are going to evaluate the accuracy of our cost model. For this purpose, we created indexes on real data containing 4, 8, and 16-dimensional Fourier vectors (derived from CAD data, normalized to the unit hypercube). The size of the databases was 2.5 and 1.0 MBytes. Figure 11 shows the results of these experiments (total elapsed time in seconds), revealing the three different cases of query processing mentioned in the beginning of this section. The

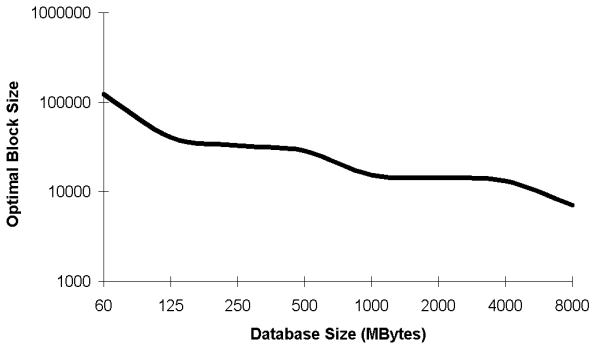


Figure 10: Optimal Block Size for Varying Sizes of the Database

left side shows the results of the 4-dimensional data space, where normal index based query processing with small block sizes is optimal. To the right, we have the high-dimensional case, where the infinite block size (sequential scan) is optimal. The interesting case (8-dimensional data space) is presented in the middle. Here, the performance forms an optimum at a block size of 32KBytes, which is very unusual in database applications. The query processing software using this logical block size (16 contiguous pages of the operating system) yields substantial performance improvements over the normal index (258%) as well as over the sequential scan (205%). The maximal speedup reached in this series of experiments was 528% over normal index processing and 500% over the infinite block size (sequential scan).

6. Conclusions

In this paper, we propose a new analytical cost model for nearest neighbor queries in high-dimensional spaces. The model is based on recent insights into the effects occurring in high-dimensional spaces and can be applied to optimize the processing of nearest neighbor queries. One important application is the determination of an optimal block size depending on the dimensionality of the data and the database size. Since the linear scan can be seen as a special configuration of an index structure with an infinite block size, an index structure using the optimal block size will perform in very high dimensions as good as a linear scan of the database. For a medium dimensionality, the optimal block size leads to significant performance improvements over the index-based search and the linear scan. An experimental evaluation shows that an index structure using the optimal block size outperforms an index structure using the normal block size of 4 KBytes by up to 528%.

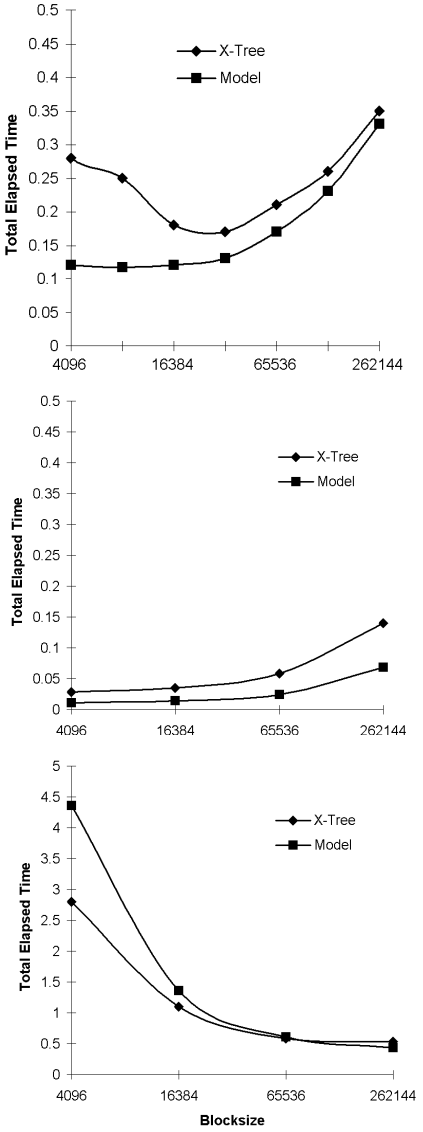


Figure 11: Accuracy of model (real data)

References

[1] Altschul S. F., Gish W., Miller W., Myers E. W., Lipman D. J.: *'A Basic Local Alignment Search Tool'*, Journal of Molecular Biology, Vol. 215, No. 3, 1990, pp. 403-410.

[2] Beckmann N., Kriegel H.-P., Schneider R., Seeger B.: *'The R*-tree: An Efficient and Robust Access Method for Points and Rectangles'*, Proc. ACM SIGMOD Int. Conf. on Management of Data, Atlantic City, NJ, 1990, pp. 322-331.

[3] Berchtold S., Böhm C., Keim D., Kriegel H.-P.: *'A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space'*, Proc. ACM PODS Int. Conf. on

Principles of Databases, Tucson, Arizona, 1997.

[4] Berchtold S., Böhm C., Braunmüller B., Keim D., Kriegel H.-P.: *'Fast Parallel Similarity Search in Multimedia Databases',* Proc. ACM SIGMOD Int. Conf. on Management of Data, Tucson, Arizona, 1997.

[5] Berchtold S., Keim D. A.: *'High-dimensional Index Structures: Database Support for Next Decades's Applications'*, Tutorial, Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, p. 501.

[6] Berchtold S., Keim D., Kriegel H.-P.: *'The X-tree: An Index Structure for High-Dimensional Data'*, 22nd Conf. on Very Large Databases, 1996, Bombay, India.

[7] Berchtold S., Keim D., Kriegel H.-P.: *'Fast Searching for Partial Similarity in Polygon Databases',* VLDB Journal, Dec. 1997.

[8] Ciacia P., Patella M., Zezula P.: *'A Cost Model for Similarity Queries in Metric Spaces'*, Proc. ACM PODS Int. Conf. on Principals of Databases, Seattle, WA, 1998, pp. 59-68.

[9] Cleary J. G.: *'Analysis of an Algorithm for Finding Nearest Neighbors in Euclidean Space',* ACM Transactions on Mathematical Software, Vol. 5, No. 2, June 1979, pp.183-192.

[10] Faloutsos C., Barber R., Flickner M., Hafner J., et al.: *'Efficient and Effective Querying by Image Content',* Journal of Intelligent Information Systems, 1994, Vol. 3, pp. 231-262.

[11] Friedman J. H., Bentley J. L., Finkel R. A.: *'An Algorithm for Finding Best Matches in Logarithmic Expected Time'*, ACM Transactions on Mathematical Software, Vol. 3, No. 3, September 1977, pp. 209-226.

[12] Hjaltason G. R., Samet H.: *'Ranking in Spatial Databases'*, Proc. 4th Int. Symp. on Large Spatial Databases, Portland, ME, 1995, pp. 83-95.

[13] Katayama N., Satoh S.: *'The SR-Tree: An Index Structure for High-Dimensional Nearest Neighbor Queries',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997.

[14] Kukich K.: *'Techniques for Automatically Correcting Words in Text'*, ACM Computing Surveys, Vol. 24, No. 4, 1992, pp. 377-440.

[15] Jagadish H. V.: *'A Retrieval Technique for Similar Shapes',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 208-217.

[16] Lin K., Jagadish H. V., Faloutsos C.: *'The TV-tree: An Index Structure for High-Dimensional Data'*, VLDB Journal, Vol. 3, 1995, pp. 517-542.

[17] Mehrotra R., Gary J. E.: *'Feature-Based Retrieval of Similar Shapes'*, Proc. 9th Int. Conf. on Data Engineering, Vienna, Austria, 1993, pp. 108-115.

[18] Mehrotra R., Gary J. E.: *'Feature-Index-Based Similar Shape Retrieval'*, Proc. of the 3rd Working Conf. on Visual Database Systems, March 1995.

[19] Roussopoulos N., Kelley S., Vincent F.: *'Nearest Neighbor Queries',* Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71-79.

[20] Shawney H., Hafner J.: *'Efficient Color Histogram Indexing',* Proc. Int. Conf. on Image Processing, 1994, pp. 66-70.

[21] Shoichet B. K., Bodian D. L., Kuntz I. D.: *'Molecular Docking Using Shape Descriptors',* Journal of Computational Chemistry, Vol. 13, No. 3, 1992, pp. 380-397.

[22] Sproull R.F.: *'Refinements to Nearest Neighbor Searching in k-Dimensional Trees',* Algorithmica 1991, pp. 579-589.

[23] Wallace T., Wintz P.: *'An Efficient Three-Dimensional Aircraft Recognition Algorithm Using Normalized Fourier Descriptors'*, Computer Graphics and Image Processing, Vol. 13, pp. 99-126, 1980.

[24] Weber R., Schek H.-J., Blott S.: '*A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces'*, Proc. Int. Conf. on Very Large Databases, New York, 1998.

[25] White, D., Jain R.: *'Similarity Indexing with the SS-Tree',* Proc. 12th Int. Conf. on Data Engineering, New Orleans, LA, 1996, pp. 516-523.