A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space

Stefan Berchtold
University of Munich
Germany
berchtol@informatik.uni-muenchen.de

Christian Böhm
University of Munich
Germany
boehm@informatik.uni-muenchen.de

Daniel A. Keim
University of Munich
Germany
keim@informatik.uni-muenchen.de

Hans-Peter Kriegel
University of Munich
Germany
kriegel@informatik.uni-muenchen.de

Abstract

In this paper, we present a new cost model for nearest neighbor search in high-dimensional data space. We first analyze different nearest neighbor algorithms, present a generalization of an algorithm which has been originally proposed for Quadtrees [13], and show that this algorithm is optimal. Then, we develop a cost model which - in contrast to previous models - takes boundary effects into account and therefore also works in high dimensions. The advantages of our model are in particular: Our model works for data sets with an arbitrary number of dimensions and an arbitrary number of data points, is applicable to different data distributions and index structures, and provides accurate estimates of the expected query execution time. To show the practical relevance and accuracy of our model, we perform a detailed analysis using synthetic and real data. The results of applying our model to Hilbert and X-tree indices show that it provides a good estimation of the query performance, which is considerably better than the estimates by previous models especially for high-dimensional data.

Key Words: Nearest Neighbor Search, Cost Model, Multidimensional Searching, Multidimensional Index Structures, High-Dimensional Data Space

1. Introduction

In this paper, we describe a cost model for nearest neighbor queries in high-dimensional space. Nearest neighbor queries are very important for many applications. Examples include multimedia indexing [9], CAD [17], molecular biology (for the docking of molecules) [24], string matching [1], etc. Most applications use some kind of feature vector for an efficient access to the complex original data. Examples of feature vectors are color histograms [23], shape descriptors [16, 18], Fourier vectors [26], text descriptors [15], etc. Nearest neighbor search on the high-dimensional feature vectors may be defined as follows:

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

$$NN(Q) = \{\bar{e} \in DS \mid \forall e \in DS\colon \|\bar{e} - Q\| \le \|e - Q\|\}\,.$$

Usually nearest neighbor queries are executed using some kind of multidimensional index structure such as k-d-trees, R-trees, Quadtrees, etc. In section 2, we discuss the different nearest neighbor algorithms proposed in the literature. We present a generalization of an algorithm, which has been originally proposed for Quadtrees [13] and show that this algorithm is optimal.

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 important not only for a theoretic complexity analysis of the average query execution time but it is also crucial for optimizing the parameters of the index structures (e.g., the block size) and for query optimization. An adequate cost model should work for data sets with an arbitrary number of dimensions and an arbitrary number of data points, it should be applicable to different data distributions and index structures, and most important, it should provide accurate estimates of the expected query execution time.

Unfortunately, existing models fail to fulfill these requirements. In particular, none of the models provides accurate estimates for nearest neighbor queries in high-dimensional space, and most models pose awkward and unrealistic requirements on the number of necessary data points preventing the models from being practically applicable. One of the reasons for the problems of existing models is that basically none of them accounts for boundary effects, i.e. effects that occur if the query processing reaches the border of the data space. As we will show later, boundary effects play an important role in processing nearest neighbor queries in high-dimensional space. Our model determines the expected number of page accesses when performing a nearest neighbor query by intersecting all pages with the minimal sphere around the query point containing the nearest neighbor. In contrast to previous approaches, our cost model considers boundary effects and therefore also provides accurate estimates for the high-dimensional case. Furthermore, our model works for an arbitrary number of data points and is applicable to a wide range of index structures such as k-d-trees, R-trees, quadtrees, etc.

Besides describing our cost model, we provide a detailed experimental evaluation showing the accuracy and practical relevance of our model. In our experiments, we use artificial as well as real data and compare the model estimates with the actually measured page counts obtained from two different index structures: the Hilbert-index and the X-tree.

2. Algorithms for Nearest Neighbor Search

In the last decade, a large number of algorithms and index structures have been proposed for nearest neighbor search. In the following, we give an overview of these algorithms.

2.1 Known Algorithms

Algorithms for nearest neighbor search may be divided into two major groups: partitioning algorithms and graph-based algorithms. Partitioning algorithms partition the data space (or the actual data set) recursively and store information about the partitions in the nodes. Graph-based algorithms precalculate some nearest neighbors of points, store the distances in a graph and use the precalculated information for a more efficient search. Examples for such algorithms are the RNG* algorithm of Arya [2] and algorithms using Voronoi diagrams [20]. Although in this paper we concentrate our discussion on partitioning algorithms, we believe that our results are applicable to graph-based algorithms, as well.

A rather simple partitioning algorithm is the bucketing algorithm of Welch [27]. The algorithm divides the data space into identical cells and stores the data objects inside a cell in a list which is attached to the cell. During nearest neighbor search the cells are visited in order of their distance to the query point. The search terminates if the nearest point which has been determined so far is nearer than any cell not visited yet. Unfortunately, the algorithm is not efficient for high-dimensional or real data. A more practical approach is the k-d-tree algorithm of Friedmann, Bentley and Finkel [12]. In contrast to Welch's algorithm, the order in which the k-d-algorithm visits the partitions of the data space is determined by the structure of the k-d-tree. Ramasubramanian and Paliwal [21] propose an improvement of the algorithm by optimizing the structure of the k-d-tree.

```
initialize PartitionList with the
    subpartitions of the root-partition
sort PartitionList by MINDIST;
while (PartitionList is not empty)
    if (top of PartitionList is a leaf)
       find nearest point NNC in leaf;
       if (NNC closer than NN)
          prune PartitionList with NNC;
          let NNC be the new NN
    else
       replace top of PartitionList with
              its son nodes;
    endif
    resort PartitionList by MINDIST;
endwhile
output NN;
```

Figure 1: Algorithm *NN-opt*

Roussopoulos et.al. [22] propose a different approach using the R*-tree [4] for nearest neighbor search. The algorithm traverses the R*-tree and stores for every visited partition a list of subpartitions ordered by their *minmaxdist*. The *minmaxdist* of a partition is the maximal possible distance from the query point to the nearest data point inside the partition. If a point is found having a distance smaller than the nearest point determined so far, all partition lists can be pruned because all nodes with a larger *minmaxdist* cannot contain the nearest neighbor. A problem of the R*-tree algorithm is that it traverses the index in a depth-first fashion. Subnodes are sorted before descent, but once a branch has been chosen, its processing has to be completed, even if sibling branches appear more likely to contain the NN. The algorithm therefore accesses more partitions than actually necessary.

In [13], Hjaltason and Samet propose an algorithm using PMR-Quadtrees. In contrast to the algorithm of Roussopoulos et.al., partitions are visited ordered by their *mindist*. The *mindist* of a partition is the minimal distance from the query point Q to any point p inside the partition P. More formally:

$$MINDIST(P, Q) = \min_{p \in P}(\|p - Q\|) .$$

The algorithmic principle of the method of Hjaltason and Samet can be applied to any hierarchical index structure which uses recursive and conservative partitioning. In Figure 1, we present a generalization of the algorithm which works for any hierarchical index structure. Pruning the partition list with a point *NNC* means that all partitions in the list which have a *mindist* larger than the distance of *NNC* to the query point are removed from the list.

2.2 Optimality of Algorithm *NN-opt*

In this section, we show that the algorithm *NN-opt* (cf.Figure 1) is optimal. For this purpose, we need to define the minimal sphere around the query point containing the nearest neighbor:

Definition 1: (NN-sphere)

Let Q be a query point and *NN* be the nearest neighbor of Q. Then $NN\text{-}dist = \|Q - NN\|$ is the distance of the nearest neighbor and the query point. The NN-sphere $SP(Q, r)$ of a query point Q is defined as the sphere with center Q and radius r =*NN-dist*.

Definition 2: (Optimality)

An algorithm for nearest neighbor search is optimal if the pages accessed by the algorithm during the nearest neighbor search are exactly the pages that intersect the NN-sphere.

Note that we use the term 'Optimality' relative to an underlying index structure and not relative to the nearest neighbor problem itself.

Lemma 1:

Algorithm *NN-opt* is an optimal algorithm according to definition 2, i.e. algorithm *NN-opt* accesses exactly the partitions which intersect the NN-sphere but no other partitions.

Proof:

From the correctness of algorithm *NN-opt* as provided in [13] it follows that any partition intersecting the NN-sphere is accessed during the search process.

To show the minimality of the accessed partitions, let us assume that algorithm *NN-opt* accesses a partition *NA*, which does not intersect the NN-sphere, i.e. $mindist(NA) > r$. Let NP_0 be the partition (data page) containing the nearest neighbor, NP_1 be the partition containing NP_0, ..., and NP_k be the partition in the root-page containing NP_0, ..., NP_{k-1}. Thus,

$$r \geq mindist(NP_0) \geq \ldots \geq mindist(NP_k) .$$

Consequently,

$$mindist(NA) > r \geq mindist(NP_0) \geq mindist(NP_k) .$$

Since NP_k is in the root-page, NP_k is replaced during the search process by NP_{k-1} and so on, until NP_0 is loaded. If, as assumed, the algorithm accesses *NA, NA* has to be on top of the partition list at some point during the search. Since $mindist(NA)$ is smaller than the *mindist* of any partition containing the nearest neighbor, *NA* cannot be loaded until NP_0 has been loaded. If NP_0 is loaded, however, the algorithm prunes all partitions which have a *mindist* smaller than r. Therefore, NA is pruned and not accessed which is in contradiction to the assumption. ■

d	number of dimensions
N	number of data points
C_{eff}	average number of data points per index page
a	edge length of a data page
NP_i	partition of the index structure containing partitions $NP_1, \ldots, NP_{i-1}$
Q	query point
DS	data space
$SP^d(E, r)$	d-dimensional hypersphere with center E and radius r
$Vol^d_{Sp}(r)$	volume of a d-dimensional hypersphere
$Vol^d_{avg}(r)$	average volume of a d-dimensional hypersphere, boundary effects considered
$Vol^d_{Mink}(r)$	Minkowski sum of an index page and a query sphere with radius r
$p(r)$, $P(r)$	distribution function of the radius, density function of the radius
NN-dist, E(NN-dist)	nearest neighbor distance, expected nearest neighbor distance
#pages, E(#pages)	number of page accesses, expected number of page accesses

3. The Cost Model

The objective of our cost model is to provide accurate estimates of the execution time of nearest neighbor queries including high-dimensional data. It is a well-known fact that simple queries, including nearest neighbor queries, are I/O-bound and only complex queries such as the spatial join may be CPU-bound. Therefore, it is justified to take the number of page accesses as a measure for the query performance. Our cost model may be used for optimizing the parameters of the index structures such as the block size as well as for query optimization.

3.1 Previous Approaches and their Problems

Due to the high practical relevance of nearest neighbor queries, cost models for estimating the number of necessary page accesses have been proposed already several years ago. The first approach is the well-known cost model proposed by Friedman, Bentley and Finkel [12]. The assumptions of the model, however, are unrealistic for nearest neighbor queries on high-dimensional data, since N is assumed to converge to infinity and boundary effects are not considered. The model by Cleary [7] extends the Friedman, Bentley and Finkel model by allowing non-rectangular-bounded pages, but still does not account for boundary effects. Sproull [25] 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. According to [25], boundary effects significantly contribute to the costs unless the following condition holds:

$$N \;>>\; C_{eff} \cdot \left(\frac{1}{\sqrt[d]{C_{eff} \cdot Vol_{Sp}^{d}\left(\frac{1}{2}\right)}} + 1 \right)^{d}$$

where $Vol_{Sp}^{d}(r)$ is the volume of a hypersphere with radius r which can be computed as

$$Vol_{Sp}^{d}(r) \;=\; \frac{\sqrt{\pi}^{d}}{\Gamma(d/2+1)} \cdot r^{d}$$
$$\text{with } \Gamma(x+1) \;=\; x \cdot \Gamma(x)\,,$$
$$\Gamma(1) \;=\; 1 \text{ and}$$
$$\Gamma\left(\frac{1}{2}\right) \;=\; \sqrt{\pi}\,.$$

Unfortunately, the assumptions made in the existing models do not hold in the high-dimensional case. The main reason for the problems of the existing models is that they do not account for boundary effects. Boundary effects is short for an exceptional performance behavior, when the query reaches the boundary of the data space. As we show later, boundary effects occur frequently in high-dimensional data spaces and lead to a pruning of major amounts of empty search space, which is not considered by the existing models. To examine these effects, we performed experiments to compare the necessary page accesses with the model estimates. Figure 2 shows the real page counts versus the estimates of the Friedman, Bentley and Finkel model. For high-dimensional data, the model completely fails to estimate the number of page accesses.

Papadopoulos and Manolopoulos present in a very recent work [19] an analysis of nearest neighbor queries using R-trees. In a recent paper [3], Arya, Mount, and Narayan develop a model that is capable of accounting for boundary effects. The problem of the Arya approach, however, is that the model still assumes N to be growing exponentially with the dimension and it also uses the L_{∞} metric, which is not suitable for most database applications. Note that our model also confirms the earlier results of Yao and Yao [28].

3.2 Overview of our Cost Model

The main objective of this paper is to present a new cost model for nearest neighbor queries in high dimensions. In contrast to existing models, our cost model provides accurate estimates of the number of page accesses in the high-dimensional case since it accounts for boundary effects. Furthermore, our model is based on the optimal algorithm for nearest neighbor search (cf. subsection 2.2) and works for an arbitrary number of data points. For the presentation of our cost model, we first assume that the data is uniformly distributed and that the split is performed in a k-d tree fashion. We will show later that our model is also applicable to arbitrary data distributions and a wide range of index structures such as k-d-trees, R-trees, quadtrees, Z-indices, etc.

The goal of our model is to determine the expected number of pages which have to be accessed in performing a nearest neighbor query. The number of data pages which have to be accessed can be determined by intersecting all pages with the minimal sphere around the query point containing the nearest neighbor. The first step in developing the cost function is to determine the average portion of a query sphere with a given query radius, which is inside the data space. Note, that the data space is assumed to be normalized to the unit hypercube $[0...1]^{d}$. Then, we determine the expected radius of the sphere, which can be described as a stochastic variable. Taking boundary effects into account, we derive the distribution function, probability density, and expected value of the nearest neighbor distance (cf. subsection 3.3). In the next step, we have to determine the number of pages intersected by the query sphere. For this purpose, we require the Minkowski sum of the query sphere and the shape of an index page (e.g., the bounding box of the page in case of the R-tree). Due to boundary effects, portions of the volume of the Minkowski sum are outside of the data space, and therefore we have to introduce some modifications to the standard Minkowski sum (cf. subsection 3.4). The last step is the integration of the separate steps into the cost function. For determining the ex-

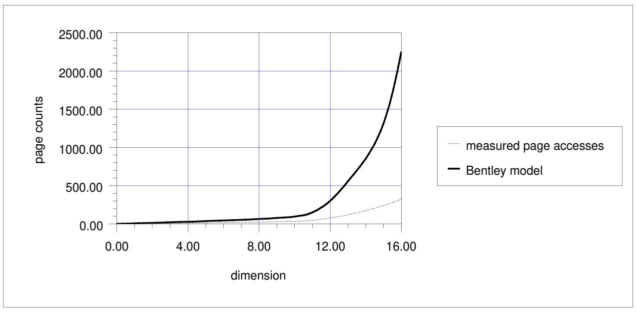

Figure 2: Real Page Counts versus Estimates by Model [12]

pected number of page accesses, we have to form the weighted average of the costs associated with the nearest neighbor distances weighted by the probability of their occurrence. The details are provided in subsection 3.5.

3.3 Expected Nearest Neighbor Distance

The goal of this subsection is to determine the expected distance between a query point and its nearest neighbor in a database of N points. Before we are able to solve this problem, however, we first consider a simpler problem, namely the expected distance of two uniformly distributed points (one query point and one data point) in the data space. Let us first assume that the data point (data entry E) has a fixed position $E = [e_1, e_2, ..., e_d]$. Then, the probability that the distance from the query point $Q = [q_1, q_2, ..., q_d]$ is less than r can be modeled as the volume of the hypersphere around E with radius r. If point E is close to the border of the data space [$\exists i \in \{1 \dots d\}: (r > e_i) \vee (e_i > 1 - r)$], we have to consider that part of the hypersphere volume is outside of the data space and does not contribute to the probability. The volume of the intersection of the data space and the hypersphere can be expressed as the integral of a piecewise defined function integrated over all possible positions of Q

$$Vol(SP^d(E, r) \cap DS) = \iint_{DS} \left(\begin{cases} 1 & if \ \|E - Q\| \le r \\ 0 & otherwise \end{cases} \right) dQ$$

$$\text{where } \|E - Q\| \le r \iff \sum_{i=1}^{d} (e_i - q_i)^2 \le r^2$$

$$\text{and } \iint_{DS} f(X) dX = \underbrace{\int_0^1 \dots \int_0^1}_{d} f(x_1, \dots, x_d) dx_1 \dots dx_d .$$

If we assume that the data point is also randomly taken from the data space, the above formula has to be averaged over all possible positions of E[1]

$$Vol_{avg}^d(r) = \iint_{DS} Vol(SP^d(E, r) \cap DS) \ dE \ .$$

Note that $Vol_{avg}^d(r)$ corresponds to the probability $P(\|E - Q\| \le r)$.

To determine the expected distance between a query point and its nearest neighbor in a database of N points, we have to determine the probability distribution of the minimum distance between query and data points. The probability that the nearest neighbor distance is at most r can also be described by the opposite: None of the N data points is in the intersection of DS and the NN-sphere. The corresponding distribution function $P(r)$ is therefore:

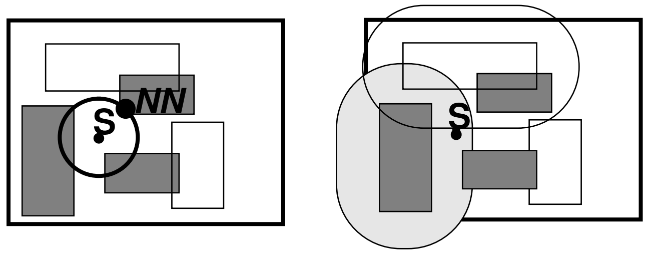

Figure 3: Transforming a Spherical Query into a Point Query by the Concept of Minkowski Sum

$$P(r) = 1 - (1 - Vol_{avg}^d(r))^N .$$

The density function $p(r)$ of $P(r)$ can be derived by determining the derivative of this function

$$p(r) = \frac{d}{dr} P(r)$$

$$= \frac{d}{dr} Vol_{avg}^d(r) \cdot N \cdot (1 - Vol_{avg}^d(r))^{N-1} .$$

From this, we obtain the expected nearest neighbor distance by the integral

$$E(NN\text{-}dist) = \int_0^\infty r \cdot p(r) \ dr$$

$$= N \cdot \int_0^\infty r \cdot \frac{d}{dr}(Vol_{avg}^d(r)) \cdot (1 - Vol_{avg}^d(r))^{N-1} dr .$$

In section 4, we will show that this formula may be used to accurately predict the expected nearest neighbor distance.

3.4 Number of Pages Intersected by the Query Sphere

In this subsection, we now determine the number of pages intersected by a query sphere with a given radius r. For this purpose, we have to determine the Minkowski sum of the query sphere and the index pages. As can be seen in Figure 3, the concept of the Minkowski sum transforms a spherical query on a set of boxes into an equivalent point query on a set of enlarged objects. The Minkowski sum directly corresponds to the volume of the intersected pages. Graphically presented, the Minkowski sum describes the volume which results from moving the center of the query sphere over the surface of the bounding box of the index page (cf. Figure 4 for an ex-

1. As DS is $[0..1]^d$, the denominator of the average is 1.

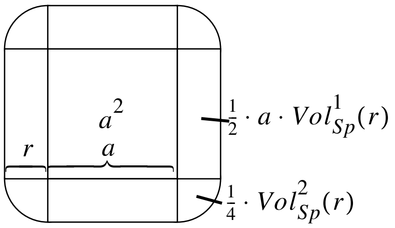

Figure 4: Example of the Minkowski Sum in Two Dimensions

ample of the two-dimensional Minkowski sum). For calculating the Minkowski sum, we have to consider volumes of each dimension between 1 and d which result from the different faces of the bounding box. If the index page is a bounding box with an extension a in all dimensions, the Minkowski sum may be calculated as

$$Vol^d_{Mink}(r) = \sum_{i=0}^{d} \binom{d}{i} \cdot a^{d-i} \cdot Vol^i_{Sp}(r) .$$

The Minkowski sum is the expected value of the hyper-volume of the bounding boxes of the data pages which are intersected by the NN-sphere. The expected value of the number of data pages can easily be determined by normalizing the Minkowski sum using the volume of the bounding box

$$\#Pages(r) = \frac{Vol^d_{Mink}(r)}{a^d} .$$

The Minkowski sum, however, does not consider boundary effects which occur in high-dimensional space because r becomes large and portions of the volume of the Minkowski sum are outside of the data space. To obtain a more realistic model for the high-dimensional case, we have to introduce some modifications to the Minkowski sum. Similar to the case described in the previous subsection, we integrate over the data space and determine the intersection of partition B with the query sphere around Q:

$$Vol^d_{MinkDS}(r) = Vol^d_{Mink}(r) \cap DS$$

$$= \iint\limits_{DS} \left(\begin{cases} 1 & if \;\; MINDIST(B,Q) \le r \\ 0 & otherwise \end{cases} \right) dr \quad .$$

If B is a rectilinear bounding box with a lower corner $[b^l_1, ..., b^l_d]$ and an upper corner $[b^u_1, ..., b^u_d]$, $MINDIST$ may be computed as

$$DIST(B,Q)^2 = \sum_{i=1}^{d} \begin{cases} 0 & if \;\; (b^l_i \le q_i \le b^u_i) \\ (b^l_i - q_i)^2 & if \;\; (q_i < b^l_i) \\ (b^u_i - q_i)^2 & otherwise \end{cases}$$

To determine the Minkowski sum according to this formula, we would need a stochastic model for the parameters b^l_i and b^u_i of the index pages. In practical experiments, we observed that in high-dimensional space usually one of the two parameters, b^l_i or b^u_i, falls together with one of the borders of the data space which results from the fact that each dimension has been split at most once. If all dimensions are of about the same significance, the split algorithm has to use all dimensions as split axes in order to obtain a high selectivity. In this case it is practically impossible in a high-dimensional space to obtain more than one split per dimension since the number of data points does not increase exponentially with the dimension. In general, the number of data points is even not high enough that all dimensions are split once. Therefore, without loss of generality, we may assume that only the first $d' \le d$ dimensions have been split at position s_i in dimension $i \; (1 \le i \le d')$. d' may be determined as

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

The Minkowski sum over all index pages which directly corresponds to the average number of pages intersected by the query sphere can be determined as

$$\#Pages(r) = \sum_{k=0}^{d'} \sum_{\{i_1, \ldots, i_k\} \in \mathcal{P}(\{1, \ldots, d'\})} Vol(SP^k([s_{i_1}, \ldots, s_{i_k}], r) \cap DS)$$

For each k, the partitions have some (d'-k)-dimensional faces inside DS. At these faces, a hyper-cylinder arises which is spherical in k dimensions (with radius r) and cubical in the remaining dimensions (with side-length 1). The spherical part may be intersected with DS and only this intersection is relevant. The second sum iterates over all elements of the power set of $\{1, ..., d'\}$and thus, selects exactly all possible k-dimensional projections of the split dimensions, encountering all possible cylinders.

For uniformly distributed data, the s_{i_j} are all at the same position $(s_{i_j} = \frac{1}{2})$. In this case, the formula becomes

$$\#Pages(r) = \sum_{k=0}^{d'} \sum_{\{i_1, \ldots, i_k\} \in \mathcal{P}(\{1, \ldots, d'\})} Vol(SP^k([\tfrac{1}{2}, \ldots, \tfrac{1}{2}], r) \cap DS)$$

As the volume of all k-dimensional cylinders is identical now, we may simplify the formula to:

$$\#Pages(r) = \sum_{k=0}^{d'} \binom{d'}{k} \cdot Vol(SP^k([\tfrac{1}{2}, \ldots, \tfrac{1}{2}], r) \cap DS) \; .$$

$$E(\#Pages) = N \cdot \int_0^{\infty} \frac{d}{dr} Vol_{avg}^{d}(r) \cdot (1 - Vol_{avg}^{d}(r))^{N-1} \cdot \sum_{k=0}^{d'} \sum_{\{i_1, \ldots, i_k\} \in \mathcal{P}(\{1, \ldots, d'\})} Vol(SP^k([s_{i_1}, \ldots, s_{i_k}], r) \cap DS) \; dr$$

Figure 5: Cost Formula for the Expected Number of Page Access

3.5 Expected Number of Page Accesses

In the previous section, we developed a model to determine the number of page accesses for a query sphere with a given radius. The goal of this section is to determine the expected number of page accesses for a nearest neighbor query.

To determine the expected number of page accesses for a nearest neighbor query, we have to integrate over the radius multiplied with the probability with which the radius occurs. More formally, the expected number of page accesses for a nearest neighbor query *E(#Pages)* may be determined as

$$E(\#Pages) = \int_0^{\infty} \#Pages(r) \cdot p(r) \; dr \; .$$

If we integrate the partial results from subsections 3.3 and 3.4, we obtain the formula presented in Figure 5.

4. Experimental Evaluation

In this section, we first describe the implementation of our cost model presented in section 3. Then, we describe the experiments conducted to show the practical applicability of our cost model and provide a short interpretation of the experimental results.

4.1 Implementation of the Cost Model

In subsection 3.3, we presented an integral formula to determine the volume of the intersection between the data space and a query sphere with radius *r*. This volume integral can be evaluated easily using numerical integration. Among the various methods, the so-called Montecarlo integration is best-suited in the high-dimensional case.

Montecarlo integration [14] is based on the principle of randomization and can be concisely described, as follows: The volume of a complex object corresponds directly to the probability that a point, randomly selected from the data space, is inside this object. Therefore, an approximation of the volume can be gained by selecting a number of points and measuring the fraction of points inside the object. Note that Montecarlo integration may be used for arbitrary data distributions.

We used a variation of this technique to determine the volume functions $Vol_{avg}^{d}(r)$ and $Vol(SP^d([\frac{1}{2}, \ldots, \frac{1}{2}], r) \cap DS)$ as well as the corresponding derivative for the required ranges of *d* and *r*. These functions are independent from individual parameters such as the number of points in the database or the capacity or geometrical shape of the data pages and are thus universally applicable for all subsequent cost computations.

The expected value of the *NN*-distance can then be efficiently integrated from the precomputed function $Vol_{avg}^{d}(r)$ by the extended trapezoidal rule. The same applies for the cost function.

4.2 Experiments

To show the accuracy of our model, we made several experiments on both, synthetic and real data. We integrated the algorithm *NN-opt* in an implementation of the well-known Hilbert-index [11] and in the original implementation of the X-tree [5]. The Hilbert-index maps *d*-dimensional points to a one-dimensional space which is then indexed by a B^+-tree. According to subsection 2.1, the algorithm *NN-opt* first examines the partition (given by a range of Hilbert values) with the lowest *MINDIST* during the search process. The X-tree is an R-tree-like multidimensional index structure which has been especially designed for indexing high-dimensional data.

Our cost model is based on an estimation of the radius of the NN-sphere. To show the accuracy of our model, we compare the average nearest neighbor distance of a uniformly distributed data set with the radius estimated by our model. For the experiments, we varied *d* from 2 to 16 using up to 369,000 data points. We averaged the radius over 100 NN-queries and found our expected nearest neighbor distance perfectly confirmed (cf. Figure 6).

To evaluate the accuracy of our cost function and its applicability to various index structures, we performed several experiments. In the first experiment, we fixed the dimension to 16 and varied the number of uniformly distributed data points from 93,000 to 2,976,000. In this experiment, we used the Hilbert index with a B^+-tree page size of 32 KBytes which implies an effective capacity of 360 data objects per data

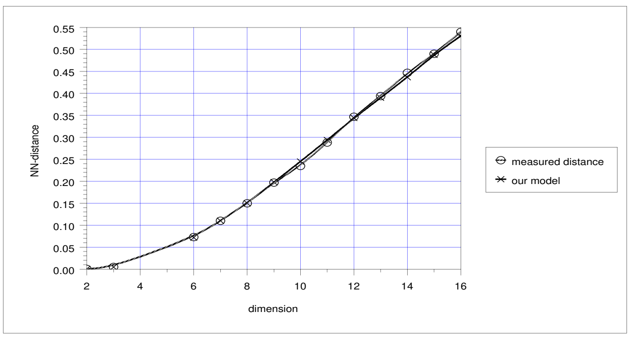

Figure 6 Expected NN-distance Depending on the Dimension

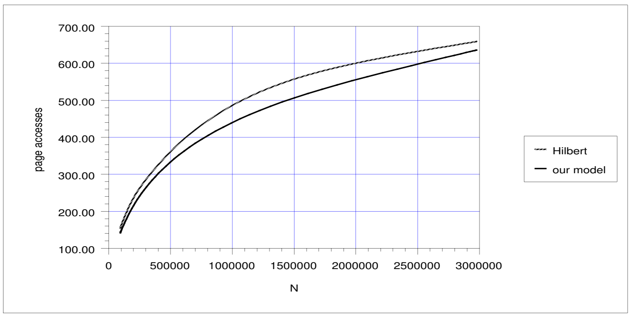

Figure 7: Expected Number of Page Accesses and Hilbert Index Performance Depending on the Number of Data Points

page. The experiment confirmed our cost model up to a relative error of 5-8% (cf. Figure 7). This remaining error is due to the impact of the specific split behavior, which is difficult to include in any formal model.

In the experiment shown in Figure 8, we compare our cost model to the performance of the X-tree with a fixed number of data pages and varying dimensionality (d = 2 .. 50). The performance of the X-tree is slightly better than the estimate of our cost model. The reason for the better performance is that the X-tree ignores 'dead space', i.e. parts of the data space which are not covered by any partition. As the experiments show, however, the estimates of our model are sufficiently close to the real performance of the X-tree. Even for low and medium dimensions, the accuracy of our model is much better than the model of Friedman, Bentley and Finkel. Note that in general our model is also applicable to R-tree-like index structures — especially in higher dimensions.

To show the practical relevance of our approach, we also performed experiments using real data. The test data used for the experiments originate from a real database consisting of high-dimensional Fourier points. Each 16-dimensional Fourier point corresponds to a region of a CAD-model describing an industrial part. We stored the Fourier-points in the Hilbert-index and performed 100 random nearest neighbor queries. Since in general the actual dimensionality of a real data set is lower than the formal dimensionality [10], we have to use the fractal dimension of the Fourier database for d in our model.

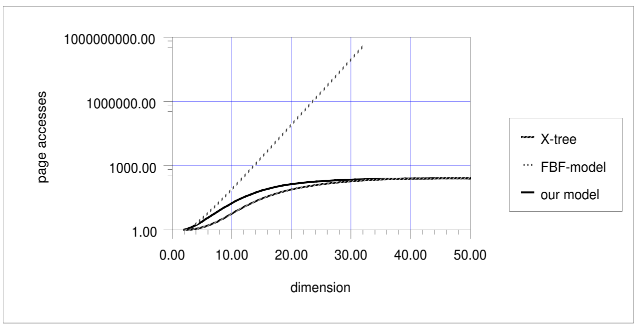

Figure 8: Expected Number of Page Accesses and Measured X-tree Performance Depending on the Dimension

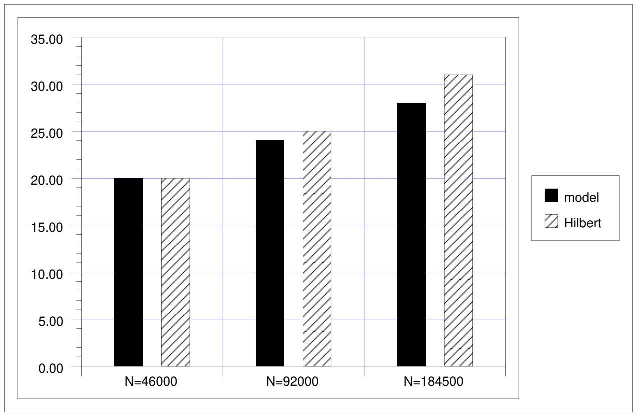

Figure 9: Application of the Cost Model to Real Data

We therefore determined the fractal dimension of the Fourier data set which is 10.56. Using 10 as the dimension in our model, we get an accurate estimation of the page accesses. Figure 9 shows the result of some experiments using different numbers N of data items.

5. Conclusion

In this paper, we presented a new cost model for nearest neighbor queries in high-dimensional data space using conservative recursive index structures such as the R-tree, k-d-B-tree or quadtree. Our cost model is accurate even in high dimensions, where other models completely fail, because our model considers boundary effects. As a further advantage, our model uses the Euclidean metric which is relevant to many database applications. We showed the applicability and accuracy of our model by presenting the results of various experiments both on synthetic and real data sets comparing our predictions with the performance of X-tree and Hilbert-based indices. Whereas previous models such as the model by Friedman, Bentley and Finkel overestimate the cost by orders of magnitude in high dimensions, our model is exact up to a moderate relative error. Our further research will focus on the extension of our model to k-nearest neighbor queries. In addition, we plan to perform a theoretically well-founded analysis of various index structures for high-dimensional 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] Arya S.: 'Nearest Neighbor Searching and Applications', Ph.D. thesis, University of Maryland, College Park, MD, 1995.

[3] Arya S., Mount D. M., Narayan O.: 'Accounting for Boundary Effects in Nearest Neighbor Searching', Proc. 11th Annual Symposium on Computational Geometry, Vancouver, Canada, 1995, pp. 336-344.

[4] 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.

[5] 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.

[6] Berchtold S., Keim D., Kriegel H.-P.: 'Fast Searching for Partial Similarity in Polygon Databases', accepted for publication: V LDB Journal, 1996.

[7] 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.

[8] Eastman C.M.: 'Optimal Bucket Size for Nearest Neighbor Searching in k-d Trees', Information Processing Letters Vol. 12, No. 4, 1981.

[9] 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.

[10] Faloutsos C., Gaede V.: 'Analysis of n-Dimensional Quadtrees Using the Hausdorff Fractal Dimension', Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996.

[11] Faloutsos C., Roseman S.: 'Fractals for Secondary Key Retrieval', Proc. 8th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, 1989, pp. 247-252.

[12] 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.

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

[14] Kalos M. H., Whitlock P. A.: 'Monte Carlo Methods', Wiley, New York, 1986.

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

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

[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] Papadpoulos A., Manolopoulos Y.:'Performance of Nearest Neighbor Queries in R-Trees', Proc. of the 6th International Conference on Database Theory, Delphi, Greece, 1997, LNCS 1186, pp. 394-408.

[20] Preparata F.P., Shamos M. I.: 'Computational Geometry', Chapter 5 ('Proximity: Fundamental Algorithms'), Springer Verlag New York, 1985, pp. 185-225.

[21] Ramasubramanian V., Paliwal K. K.: 'Fast k-Dimensional Tree Algorithms for Nearest Neighbor Search with Application to Vector Quantization Encoding', IEEE Transactions on Signal Processing, Vol. 40, No. 3, March 1992, pp. 518-531.

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

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

[24] 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.

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

[26] 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.

[27] Welch T.: 'Bounds on the Information Retrieval Efficiency of Static File Structures', Technical Report 88, MIT, June 1971.

[28] Yao A.C., Yao F.F.:'A General Approach to D-Dimensional Geometric Queries', Proc. of the ACM Symposium on Theory of Computing, 1985