A Framework for Accelerating High-dimensional NN-queries

Christian A. Lang
clang@cs.ucsb.edu

Ambuj K. Singh
ambuj@cs.ucsb.edu

Technical Report TRCS01-04

Department of Computer Science
University of California
Santa Barbara, CA 93106

Abstract

The performance of nearest neighbor (NN) queries degrades noticeably with increasing dimensionality of the data. This stems not only from reduced selectivity of high-dimensional data but also from an increased number of seek operations during query execution.

We propose a new framework to transform NN queries into at most two range queries. This is achieved by first estimating the NN-radius, performing a range query with this radius, and (if not enough NNs were found) estimating the NN-radius again and performing a second range query. Since range queries know the set of pages to be read in advance, a page read scheduler can be employed to minimize read costs. This scheme guarantees that the query cost is bounded from above by the cost of 2 linear scans over a subset of the data pages, while typically resulting in costs well below a single scan.

Our framework can be instantiated with different range query index structures, radius estimators, and page read schedulers. We present one possible implementation of the radius estimator based on sampling and the fractal dimensionality of data. Since the overall cost depends on the quality of this radius estimate, we examine it analytically and experimentally. Our analysis for uniform data indicates that the estimation error is below 14% and that errors in the fractal dimensionality estimation have only minor impact on the overall cost.

Experiments on synthetic and real datasets show that our new technique reduces the I/O cost during NN queries by a factor of up to 15 for an R*-tree based index structure and up to 5 for an X-tree-like bulkloaded index structure.

1 Introduction

Nearest neighbor queries are an important query type for high-dimensional data, such as multimedia data. The goal of such queries is to obtain the k closest data elements (points, pictures, sounds, etc.) for a given query element. More formally: given a query point q, the NN query result is a set $NN_k(q)$ of at least k points such that

$$\forall o \in NN_k(q), \forall o' \in DB - NN_k(q) : d(o,q) < d(o',q)$$

where DB denotes the full dataset and $d(o,q)$ denotes the distance between o and q.

On large databases, such queries can be very time-consuming, especially when the number of dimensions is high. Much research has focused therefore on accelerating this type of queries. In general, this can be achieved in two ways: by reducing the number of disk pages to read, or by reducing the number of disk seek operations during page reads.

An algorithm that minimizes the number of index pages to read was proposed by Hjaltason and Samet [10]. Their algorithm was proven to be optimal in the sense that only pages intersecting the NN-query sphere are accessed. However, since the pages are read according to their distance to the query point rather than their location on disk, most page reads require a disk seek operation. The cost due to this effect gets more pronounced with higher dimensionalities.

On the other hand, techniques that reduce the number of page seeks cannot guarantee a minimum number of pages to be read. One extreme example is the linear scan over the whole data file in order to determine the NNs. This approach obviously reduces the amount of seeks but causes many unnecessary page reads. Other more sophisticated techniques were proposed but the tradeoff remains the same: either the amount of page reads is reduced, or the amount of seeks is.

An algorithm that gets the best of both worlds would be reading only the pages that contain the NNs as sequentially as possible. The problem with this approach is that we do not know the required pages in advance.

Our new approach tries to get as close as possible to this hypothetical algorithm. This is achieved by first estimating the NN-query radius by inspecting the current query and dataset. Once this radius is obtained, it can be used to transform the NN-query into a range query that reads the pages as sequentially as possible (since it knows all pages in advance). In case we underestimate the query radius, a second range query with a larger range is necessary. This larger range is obtained from points read during the first range query. We can show that the second range is a (tight) upper bound on the query radius. Therefore, we never have to execute more than two range queries for a single NN query.

A major contribution of this paper is a general framework for transforming NN queries into at most two range queries. Key elements such as the NN-radius estimation algorithm, or the underlying index structure can be replaced modularly. The framework is therefore widely applicable. Furthermore, we can guarantee that the query cost is bounded from above by the cost of 2 linear scans over a subset of the data while typically staying well below the cost of a single scan.

As a second contribution, we give one possible NN-radius estimation algorithm based on sampling and the fractal dimensionality. We first determine the k-NN radius on a data sample in memory. Using the fractal dimensionality of the full dataset and the data sample, we can compute how the query radius changes due to sampling. This information can then be used to adjust the first radius estimate. This adjusted radius is the desired estimate.

Our third contribution is an extensive analysis of the I/O overhead of the framework. The analysis for uniform data indicates that the estimation error is below 14% and that errors in the fractal dimensionality estimation have only minor impact on the overall cost.

We illustrate the wide applicability of our framework by using a number of different index structures such as R*-tree [2], X-tree [7], SS/SR-tree [20, 12], and VA-file [19]. One key advantage is that index structures without NN query support, such as the Pyramid tree [5], can be extended by our framework to support this query type. As a by-product, we demonstrate how our framework can be modified to support approximate NN-queries.

Our experimental results show that our accelerated NN-query algorithm performs up to 15 times faster than the R*-tree with the optimal NN-query algorithm by Hjaltason and Samet [10], and up to 5 times faster than the X-tree. Even non-clustering index structures, such as the VA-file, are accelerated by our technique. What is surprising is that our technique outperforms the NN-query algorithms proposed as a part of these index structures.

This paper is organized as follows. Section 2 discusses our accelerated k-NN-query algorithm.

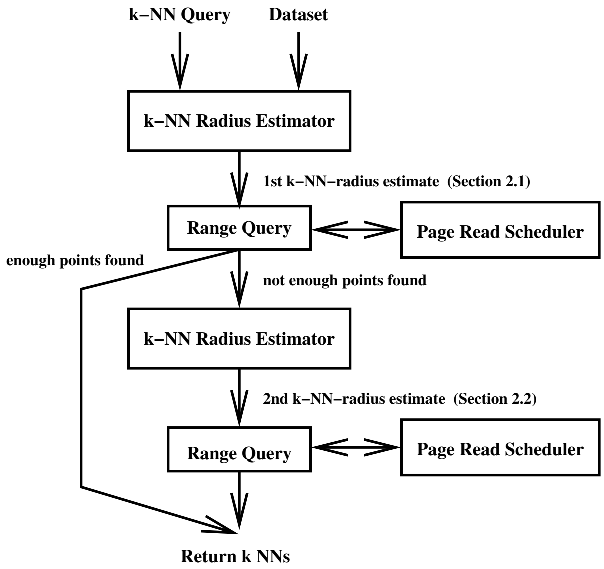


Figure 1: The accelerated NN-query framework

We compare this algorithm with a hypothetical algorithm that knows the exact query radius in advance in Section 3. In Section 4, we present our experimental results on synthetic and real datasets. Section 5 summarizes other work on accelerating queries using statistics. The conclusions can be found in Section 6.

2 Accelerated k-NN-Query Algorithm

The optimal k-NN-query algorithm [10] reads disk pages according to their distance from the query point. In high dimensions, this order is quite different from the sequential order of the pages on disk. As a result, a large fraction of the time for traditional k-NN-queries is spent in expensive seek operations. If all pages to be read during a NN query would be known in advance, these expensive seeks could be avoided.

A significant contribution of this paper is to provide a framework to achieve an optimal disk page read schedule during NN-queries. Figure 1 depicts this framework as a flowchart. For every incoming NN-query, we compute a first estimate of its NN-query radius. Using this estimate, we can perform a range query on the full dataset. Since all pages are known in advance for range queries, the amount of seek operations can be minimized by employing a specific page read scheduler. If the first estimate was too large, at least k points were in the range, and we can return the k closest ones as the result set. If the first estimate was too small, less than k points were in the range and we need to increase the search radius. This second radius estimate is then used to perform another range query. As we will show, the second radius estimator always overestimates the real NN-radius slightly, resulting in at least k points in the second range. Therefore, at most two range queries are necessary to answer the k-NN query, resulting in a worst case cost of two linear scans over a subset of the data pages (if an optimal page read scheduler is used).

Note that our framework is based only on information about the dataset and the current query. Any specific implementation of the modules (depicted with boxes in the figure) can be used. The only restriction we make is that the second radius estimator does not *under*estimate the real NN-radius. We will give a k-NN-radius estimator based on sampling in the following. It estimates

N	number of data points
σ	data sampling rate
C_{eff}	effective data page capacity
E	embedding dimensionality
D_2	Correlation fractal dimensionality of the full dataset
D'_2	Correlation fractal dimensionality of the data sample
r_{sample}	sampling-based k-NN radius estimate
r_{first}	first k-NN radius estimate
r_{second}	second k-NN radius estimate
$d_{nn}^{full}(k)$	radius estimate based on D_2 (full dataset)
$d_{nn}^{sample}(k)$	radius estimate based on D_2 (sample data)
$r_{nn}^{full}(k)$	radius estimate with boundary effects (full dataset)
$r_{nn}^{sample}(k)$	radius estimate with boundary effects (sample data)
k	number of NNs

Table 1: Notation used in the paper

the NN-radius using a sample in memory and compensates for the sampling by taking the fractal dimensionality of the dataset into account.

Note that our technique works also for other radius estimation methods (such as histograms). Similarly, we will show that our technique works for different index structures (i.e. range query algorithms). In fact, it can even be applied to add NN queries to index structures that do not offer NN-query support, such as the Pyramid tree [5].

Sections 2.1 and 2.2 discuss in more detail how the first and second radius estimate are obtained. In Section 2.3, we present an implementation of our framework using these estimators and show how this algorithm can be modified to obtain an approximate NN-query algorithm. In the discussion throughout the paper we assume that the L_∞-norm is used for distance measurements.

2.1 First Radius Estimate

In this subsection, we will show how the first radius estimator can be implemented. The estimator presented here is particularly suited for high-dimensional data, since it is based on sampling.

The basic idea behind the first radius estimator is as follows. For a given dataset, obtain a data sample that fits into memory and estimate the fractal dimensionality of the full dataset and the sample. These steps are all part of precomputation. Once a NN-query comes in, we run the same query on the sample in memory and obtain a first radius estimate. Obviously, this radius estimate is too large because of the sampling. Therefore, we compensate for this effect by taking the fractal dimensionalities into account: since we can compute the expected NN-radii for the full dataset and its sample from their fractal dimensionalities, we can also compute the expected *change* in NN-radius when moving from the full dataset to the sample and vice versa. The detailed formulas are presented next.

Assume we have a dataset of size N and we obtain an in-memory sample of size $N \cdot \sigma$; σ is called the *sampling rate* [1]. Furthermore, let q be a query point and k the number of its NNs we are looking for. We can obtain a first (rough) estimate of the query radius by computing the k NNs of q on the sample. Since the sample is stored entirely in memory, this can be done very efficiently.

[1]Table 1 summarizes all symbols used in this paper.

Let us denote this estimate by r_{sample}. Since there might be a point that is a k-NN of q but is not in the sample, r_{sample} will usually be larger than the real k-NN-radius. If we knew how the radius changes with the sampling rate, we could compensate for this change. For uniform data, this is a trivial problem since the density of the data is the same everywhere. Real data, however, is clustered and the density varies over the data space. One way of modeling real data is by considering its fractal properties. Korn et al. [14] give formulas for the k-NN-radius based on the *fractal dimensionality* of data. More precisely, they calculate the radius as follows:

$$d_{nn}(k) = \frac{1}{2} \cdot \left(\frac{k}{N-1} \right)^{\frac{1}{D_2}}$$

where D_2 is the *Correlation* fractal dimensionality of the dataset. We denote this radius by $d_{nn}^{full}(k)$ to emphasize that it is computed for the full dataset. Similarly, we can calculate the k-NN radius for the sample dataset:

$$d_{nn}^{sample}(k) = \frac{1}{2} \cdot \left(\frac{k}{N \cdot \sigma - 1} \right)^{\frac{1}{D_2'}}.$$

where D_2' denotes the Correlation fractal dimensionality of the sample dataset.

With these two radius estimates, we can now calculate the (average) rate of change in the NN-query radius due to sampling. Let us denote the value of the first radius after compensation by r_{first}. Then we have

$$r_{first} = \frac{d_{nn}^{full}(k)}{d_{nn}^{sample}(k)} \cdot r_{sample}.$$

We will discuss the quality of this estimate in Section 3.1.

The fractal dimensionalities D_2 and D_2' can be computed using a box-counting algorithm as suggested by Belussi and Faloutsos [3]. We found in our experiments, however, that this algorithm is not applicable to high-dimensional data since the point sets become too sparse. We therefore used a modified version of the box-counting algorithm as defined in [15]. Similar to the original box-counting algorithm, this algorithm has a complexity of $O(N \log N)$. Amortized over all queries, this cost is negligible.

Note that the estimator we just presented, is only one way of calculating r_{first}. Other estimators (e.g. histogram-based) are possible but not as well-suited for high-dimensional data. The reason is that with growing dimensionality, the amount of statistics that needs to be stored grows exponentially. Sampling, on the other hand, is independent of the data dimensionality.

2.2 Second Radius Estimate

The second radius estimator we need to design, can take the results of the first range query into account. Additionally, it has to ensure that its estimate is an upper bound on the real NN-radius. We describe one possible design choice in this subsection but, again, others are possible.

Let P be the set of index pages intersected by the range query with radius r_{first}. Let C_{eff} be the effective capacity of each disk page. This is usually less than the page capacity due to the reduced page utilization in index structures. Assume that $C_{eff} \cdot |P| \geq k$. Then we can simply pick q's k closest points from the pages in P and calculate the smallest radius that covers them. This second radius estimate is denoted by r_{second}. Obviously, r_{second} is larger than the real radius since the corresponding query range centered at q covers k points read from the pages, but not necessarily the k globally closest ones.

What happens if $C_{eff} \cdot |P| < k$? This means that we do not have k points after the first phase, implying that two range queries may not be enough. In practice, $k << C_{eff}$ and therefore it suffices

Input: query point q, number of nearest neighbors k
Output: k nearest neighbors of q

(1) perform a k-NN query on sample in memory; this yields a k-NN-radius r_{sample}
(2) $r_{first} := r_{sample} \cdot \frac{d_{nn}^{full}(k)}{d_{nn}^{sample}(k)}$
(3) perform a range query with radius r_{first} on full dataset;
this results in a set R of data points and a set P of index leaf pages
accessed during the query
(4) if $|R| \geq k$, return the k closest points from R, **stop**
(5) otherwise (more NNs need to be found), compute the k closest points stored in P and
calculate the smallest radius r_{second} enclosing them
(6) perform a modified range query around q with radius r_{second} on full dataset;
this results in a set R' of data points and a set P' of index leaf pages
accessed during the query
(7) return the k closest points from R', **stop**

Figure 2: The accelerated k-NN query algorithm

to access at least one page. In our implementation, we ensure that at least one page is intersected by taking the maximum between r_{first} and the distance between q and the closest sample point as the first radius estimate. Henceforth, we assume that at least k points were read during the first range query.

The next subsection shows how both radius estimates are embedded in the overall k-NN query algorithm.

2.3 k-NN Query Algorithm

In order to evaluate our accelerated query algorithm, we will instantiate our framework from now on as follows. The radius estimators are assumed to work as described in the last two subsections. The range query algorithm used is the R*-tree range query algorithm. The page read schedulers are irrelevant for the following discussions. We will instantiate them in the experimental section.

With these instantiations, our accelerated k-NN query algorithm works as follows. For any dataset and index structure, we precompute a sample of the dataset, the Correlation fractal dimensionality D_2 and D_2' of the full dataset and the sample dataset, respectively. This can be done in $O(N \log N)$ time. During runtime, for each query point q, the sample is used to calculate the first radius estimate, r_{first} as described in Section 2.1. Then, a range query around q with radius r_{first} is performed on the full dataset yielding a set of pages P that intersect the query range. If more than k points were read (r_{first} is greater than the real k-NN-radius), the algorithm returns the k closest ones and stops. Otherwise, P is used to determine the second range query radius r_{second}, as discussed in Section 2.2. Then, a second range query around q with radius r_{second} is performed yielding at least k points (r_{second} is a upper bound on the real radius). The closest k points are returned as the result. Figure 2 gives the algorithm in pseudo code. Note that only steps (3) and (6) induce disk I/O.

The range query in step (6) is modified as follows. Since the k closest points from the pages P are already known, this range query needs to read only the pages from $P' - P$. In general, both range queries use the heuristic suggested by [17] to minimize the I/O cost for reading a set of disk pages.

2.4 Approximate k-NN Query Algorithm

The following modification of our accelerated k-NN query algorithm yields an approximate k-NN query algorithm with less I/O cost. Instead of performing a second range query, we can simply report the k closest points encountered during the first range query. As we will show in Section 3.1, for typical applications, r_{first} is already close to the real radius. Figure 3 lists the modified algorithm in pseudo code.

Input: query point q, number of nearest neighbors k
Output: k approximate nearest neighbors of q

(1) perform a k-NN query on sample in memory; this yields a k-NN-radius r_{sample}
(2) $r_{first} = r_{sample} \cdot \frac{d_{nn}^{full}(k)}{d_{nn}^{sample}(k)}$
(3) perform a range query around q with radius r_{first} on full dataset; this results in a set R of data points and a set P of index leaf pages accessed during the query
(4) return the k closest points from R, **stop**

Figure 3: The approximate k-NN query algorithm

This section defined a new way of estimating the k-NN-query radius. It is achieved by first estimating the radius using a data sample and the fractal dimensionality of the dataset. Another novelty is the second radius estimator that takes already read pages (and points) into account and guarantees an overestimation of the query radius. In Section 3, we analyze the I/O cost of our NN-query algorithm and explain how errors in the radius estimations affect this cost.

3 I/O Overhead of our Technique

In this section, we analyze the I/O overhead caused by incorrect radius estimations. Section 3.1 examines the first radius estimate, Section 3.2 discusses the second estimate, and Section 3.3 analyzes the combined overhead of both phases together.

We compare our accelerated NN-query algorithm to an algorithm with an "oracle", denoted by ORACLE. This algorithm is assumed to know the exact k-NN-query radius already in advance and can therefore optimally read all necessary pages. In other words, ORACLE is obtained by instantiating the first radius estimator with a "perfect" estimator returning always the correct NN-radius. We will demonstrate that our algorithm performs nearly as good as ORACLE.

3.1 Quality of r_{first}

The quality of the first radius estimate can be expressed in several ways. We first examine the deviation of the radius estimate from the correct NN-radius for uniform datasets (Section 3.1.1). We then show that similar numbers hold for real datasets (Section 3.1.2). Since our first radius estimate is based on the fractal dimensionality D_2 of the dataset, we analyze how an error in D_2 affects the radius estimate (Section 3.1.3). Finally, we give a formula for the page reads during the first phase that can be used to compare the I/O cost of our algorithm to ORACLE (Section 3.1.4).

3.1.1 Expected Error for Uniform Data Distribution

In order to analyze the expected error in the first radius estimation analytically, let us assume a uniform data distribution and a normalized dataspace. In this setting, we can calculate the expected query radius (the radius assumed by ORACLE) as follows. With boundary effects[2], the volume of the query shape (query hypercube in the case of the L_∞-norm) depends on the location of the query point. The expected volume is therefore the average over all possible query point locations p:

Theorem 1 (Average ϵ-query volume with boundary effects)
The average volume $qvol(\epsilon)$ of an ϵ-query when taking boundary effects into account, is

$$\frac{1}{2}+\epsilon+\frac{1}{2}(\min\{1-\epsilon,\epsilon\}^2+\max\{1-\epsilon,\epsilon\}^2)+\epsilon\min\{1-\epsilon,\epsilon\}-(1+\epsilon)\max\{1-\epsilon,\epsilon\}+\begin{cases} 2\epsilon-4\epsilon^2 & \text{if } \epsilon\le\frac{1}{2} \\ 2\epsilon-1 & \text{otherwise}\end{cases}$$

Proof: The expected volume over all possible query point locations p is

$$\int_p \prod_{i=1}^{E} (\min\{p_i+\epsilon,1\}-\max\{p_i-\epsilon,0\})dp$$

where E denotes the embedding dimensionality. This is equivalent to

$$\int_p \prod_{i=1}^{E} \begin{cases} 2\epsilon & \text{if } p_i+\epsilon<1 \text{ and } p_i-\epsilon>0 \\ p_i+\epsilon & \text{if } p_i+\epsilon<1 \text{ and } p_i-\epsilon\le 0 \\ 1-p_i+\epsilon & \text{if } p_i+\epsilon\ge 1 \text{ and } p_i-\epsilon>0 \\ 1 & \text{otherwise}\end{cases} dp$$

which is equivalent to

$$\prod_{i=1}^{E} \int_{p_i} \begin{cases} 2\epsilon & \text{if } p_i<1-\epsilon \text{ and } p_i>\epsilon \\ p_i+\epsilon & \text{if } p_i<1-\epsilon \text{ and } p_i\le\epsilon \\ 1-p_i+\epsilon & \text{if } p_i\ge 1-\epsilon \text{ and } p_i>\epsilon \\ 1 & \text{otherwise}\end{cases} dp_i.$$

The integral can be calculated as

$$\int_\epsilon^{1-\epsilon} 2\epsilon\ dp_i + \int_0^{\min\{1-\epsilon,\epsilon\}} (p_i+\epsilon)\ dp_i + \int_{\max\{1-\epsilon,\epsilon\}}^{1} (1-p_i+\epsilon)\ dp_i + \int_{1-\epsilon}^{\epsilon} 1\ dp_i$$

which is

$$\frac{1}{2}+\epsilon+\frac{1}{2}(\min\{1-\epsilon,\epsilon\}^2+\max\{1-\epsilon,\epsilon\}^2)+\epsilon\min\{1-\epsilon,\epsilon\}-(1+\epsilon)\max\{1-\epsilon,\epsilon\}+\begin{cases} 2\epsilon-4\epsilon^2 & \text{if } \epsilon\le\frac{1}{2} \\ 2\epsilon-1 & \text{otherwise}\end{cases}$$

□

The average L_∞-distance of a data point to its kth nearest neighbor is the ϵ for which

$$qvol(\epsilon)^E = \frac{k}{N}$$

[2]By "boundary effect" we mean the effect that for query points close to the boundary, the query shape has to grow in order to cover the same amount of points.

Let us denote this value by $r_{nn}^{full}(k)$. Similarly, let us denote the value of ϵ for which

$$qvol(\epsilon)^E = \frac{k}{N \cdot \sigma}$$

by $r_{nn}^{sample}(k)$.

Thus, we know the expected correct query radius. However, what is the expected value of r_{first}? Since we assume uniformity, $D_2 = D_2'$ and therefore

$$\frac{d_{nn}^{full}(k)}{d_{nn}^{sample}(k)} \approx \left(\frac{N \cdot \sigma}{N}\right)^{\frac{1}{D_2}}.$$

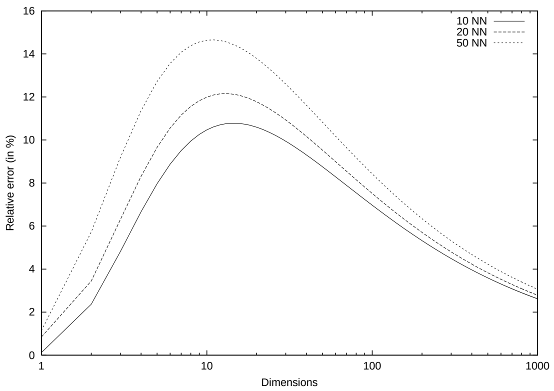
Therefore,

$$r_{first} \approx r_{nn}^{sample}(k) \cdot \sigma^{\frac{1}{D_2}}.$$

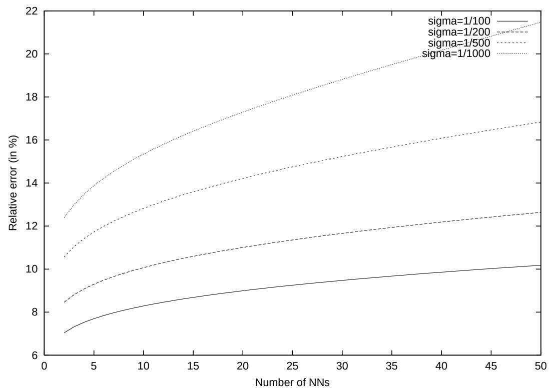
The relative error of r_{first} is then given as

$$\frac{r_{first} - r_{nn}^{full}(k)}{r_{nn}^{full}(k)}$$

which is plotted in Figure 4. Subfigure 4(a) plots the relative error as a function of the underlying



(a) Relative Error of r_{first} (uniform 100,000 points, $\sigma = \frac{1}{100}$)



(b) Relative Error of r_{first} (uniform 100,000 points, 60 dimensions)

Figure 4: Relative error of r_{first}

data dimensionality. The sampling rate σ is assumed to be $\frac{1}{100}$. For lower dimensionalities, the error is small because the estimate r_{sample} is close to the correct radius value. For higher dimensionalities, all points tend to be equidistant (see [8]). Therefore, the error in the k-NN radius estimate has to drop as well. This leaves us with a maximum at around 10 dimensions. Depending on k, this error ranges between 10% and 14%. Subfigure 4(b) plots the same relative error for varying k (number of NNs). With increasing k, the error of the radius estimate increases. However, it stays below 22% even for a sampling rate of 1/1000. This shows that (at least for uniform data) our first radius estimate is very close to the correct NN-radius.

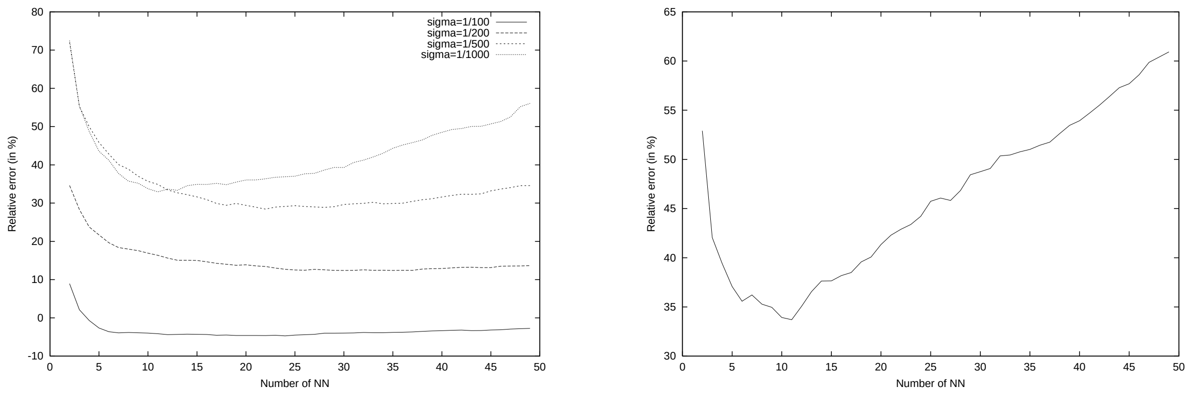
We can also see that the relative error is always positive for uniform data, meaning that r_{first} overestimates the correct radius. No second range query is required in these cases. Real data, however, can cause an underestimation as well. The next subsection gives an example for this case.

UNIFORM	100,000 8-dimensional points distributed uniformly.
TEXTURE	26,697 48-dimensional points representing texture feature vectors obtained from the Corel Image Collection (transformed using KLT).
LANDSAT	275,465 60-dimensional points representing texture feature vectors of Landsat images (transformed using KLT).

Table 2: Datasets used in the experiments

3.1.2 Measured Error for Real Data

This section shows how the relative error of r_{first} varies for real datasets (cf. Table 2 for a description of all datasets used). Figure 5 shows the results. Our first dataset, LANDSAT, contains more than a quarter million 60-dimensional points and is highly clustered. Subfigure 5(a) shows that — similar to the uniform case — with increasing k, the relative error increases slightly. The same holds for the sampling rate. The smaller the sample, the higher the relative error. However, two differences can be noted compared to the uniform case. First, for a sample rate of 1/100, the relative error drops below zero for $k > 3$. This means that the first radius underestimates the real radius, causing a second range query. We will examine the percentage of underestimations more closely in Section 3.2.



(a) Relative error of r_{first} (LANDSAT dataset)

(b) Relative error of r_{first} (TEXTURE dataset, $\sigma = 1/100$)

Figure 5: Relative error of r_{first} for real data

The second difference that can be seen from the graphs is that the error is higher for very small k and then drops quickly until it reaches a minimum. This can be explained as follows. Since the data is clustered, the distances between data points vary a lot (compared to uniform data). Since our queries are density-biased, the distance of data points from the query point varies a lot as well. If k is large, the effect of sampling is alleviated by the large number of points in the query range. If k is small, sampling can cause r_{sample} to be much larger than the real radius because the point distances have such a high variance. Even the compensation via the fractal dimensionality cannot counteract this effect because it describes the dataset globally, whereas this effect is local.

Figure 5(b) shows a similar behavior for the TEXTURE dataset. The u-shape of the curve is more pronounced and the errors are overall higher. This is because TEXTURE contains less than

30,000 points. A sample rate of $\sigma = 1/100$ leaves us with less than 300 points. The 50 NNs are 1/6 of all points! But even with such a small sample, the errors stay below 65%, and are typically below 40%.

3.1.3 Stability of r_{first} with Respect to D_2

Unless the fractal dimensionality of a dataset is known in advance (e.g. for synthetic datasets), it has to be computed in some way (e.g. using the box-counting method [14]). This computation can, however, induce some error in the fractal dimensionality. This error can again cause an error in the NN-radius estimation. In this section, we examine the impact of an error in D_2 on the quality of r_{first}.

Let $\triangle D_2$ denote the absolute error in D_2. Then

$$\sigma^{\frac{1}{D_2+\Delta D_2}} = \sigma^{\frac{1}{D_2}} \cdot \sigma^{-\frac{1}{\frac{D_2^2}{\Delta D_2}+D_2}} = \sigma^{\frac{1}{D_2}} \cdot (1 + \sigma^{-\frac{1}{\frac{D_2^2}{\Delta D_2}+D_2}} - 1).$$

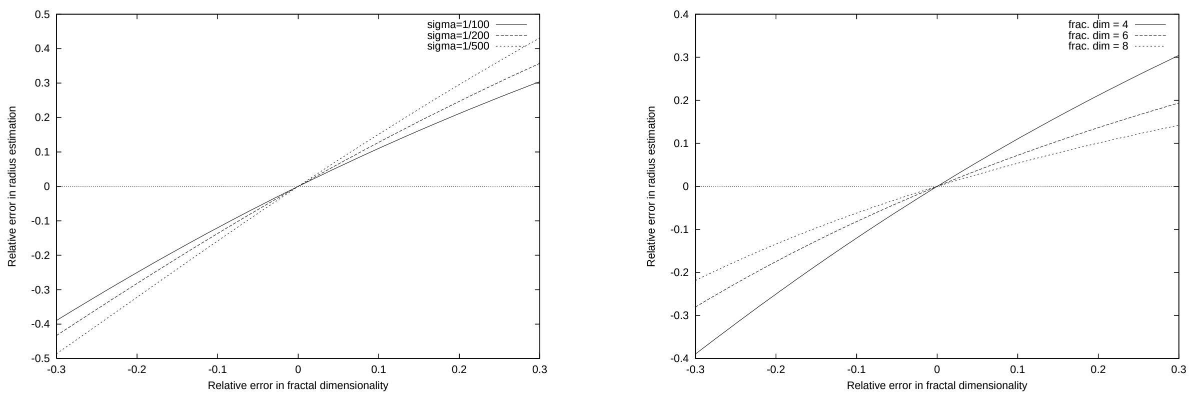
Now let $\widetilde{r_{first}}$ denote the first radius estimate with error. We have

$$\widetilde{r_{first}} = r_{sample} \cdot \sigma^{\frac{1}{D_2}} \cdot (1 + \sigma^{-\frac{1}{\frac{D_2^2}{\Delta D_2}+D_2}} - 1).$$

In other words, the relative error in the radius estimation is

$$\sigma^{-\frac{1}{\frac{D_2^2}{\Delta D_2}+D_2}} - 1.$$

Figures 6(a) and 6(b) plot this error for different numbers of NNs and different fractal dimensionalities, respectively. We chose fractal dimensionalities between 4 and 8 since these are typical values



(a) Relative Error of r_{first} (uniform 100,000 points, $D_2 = 4$)

(b) Relative Error of r_{first} (uniform 100,000 points, $\sigma = 1/100$)

Figure 6: Stability of r_{first}

for most real datasets we looked at. As can be seen from Figure 6(a), for less than 10% error in D_2, the radius estimation error is below 10% (independent of k). Actually, the dependency between the error in D_2 and the estimated radius error is nearly linear. Underestimations of D_2 cause an underestimation of the radius, and vice versa for overestimations. This can be explained as follows. If D_2 is underestimated, the effect of sampling is less pronounced. Hence, less compensation is applied to r_{sample}. On the other hand, if D_2 is overestimated, sampling will cause the NN-radius to shrink more, and therefore more compensation is applied to r_{sample}.

In our experiments on synthetic datasets (where D_2 is known), the relative error in D_2 computed with the box-counting algorithm was less than 2%. Even though it can not be known whether this holds true also for real clustered data, it is a strong indicator that the error caused by the fractal dimensionality calculation is negligible.

3.1.4 I/O Overhead of First Phase

The knowledge of the expected first radius estimate can now be used to calculate the expected number of pages that have to be read during processing of the first range query. By comparing this number to the expected number of pages read by ORACLE, we can compute the overhead caused by our estimation.

Theorem 2 (Average number of page accesses during first range query) *Assuming uniform data distribution and a normalized (unit) dataspace, the expected number $F(r)$ of page accesses during the first range query for a search radius r can be computed as*

$$F(r) = (1 + 2r)^s$$

where $s = \left\lceil \log_2(\frac{N}{C_{eff}}) \right\rceil$ is the number of split dimensions.

Proof: In the case of uniform data distribution, the index pages are created by splitting the dataspace consecutively in the middle in different dimensions until the required number of pages is reached. The number of splits required to obtain at least $\frac{N}{C_{eff}}$ pages is

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

Assuming a normalized dataspace, the probability that a page is accessed during a query is given by the cutoff volume of the Minkowski sum of the page with the query shape. "Cutoff" means that the Minkowski sum has to be truncated at the dataspace boundary.

Since such a page has extent $1/2+r$ in the split dimensions, and extent 1 in the other dimensions, its overall volume is $(\frac{1}{2}+r)^s$. This is the probability that a single page is accessed during the query. In order to get the average number of pages intersected, we have to sum up the probabilities for all pages. Since all probabilities are equal and we have 2^s pages the expected number of pages accesses is

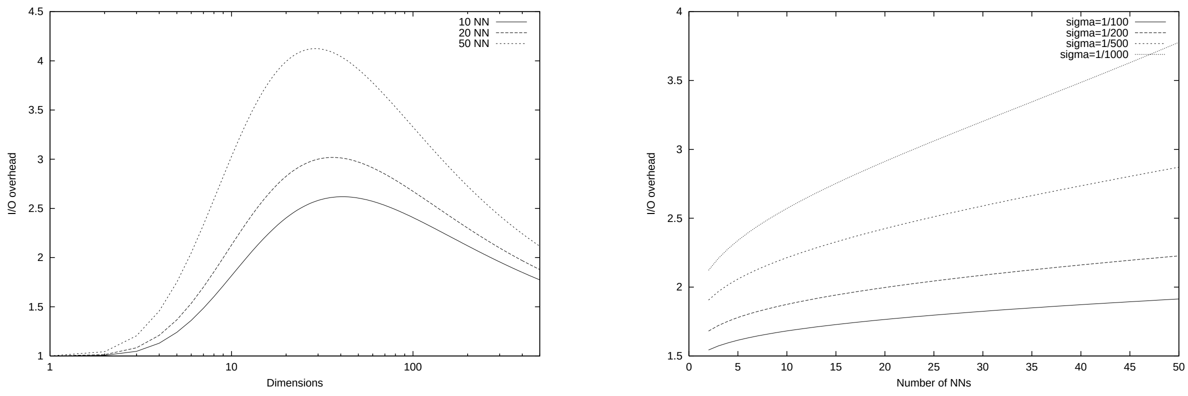
$$(\frac{1}{2} + r)^s \cdot 2^s.$$

□

Now we can compute the I/O overhead of the first phase:

$$\frac{F(r_{first})}{F(r_{nn}^{full})}.$$

It is plotted in Figure 7. Subfigure 7(a) plots the I/O overhead as a function of the underlying data dimensionality. The sampling rate σ is assumed to be $\frac{1}{100}$. Subfigure 7(b) plots the same I/O overhead for varying k. Both plots show the same behavior as the relative error plots for r_{first}



(a) I/O overhead for $\sigma = \frac{1}{100}$

(b) I/O overhead for 60 dimensional data

Figure 7: I/O overhead of first phase (uniform $100,000$ points)

(Figure 4). The I/O overhead stays between 1 and 4, depending on the sampling rate. This means that we have to read at most 4 times as many pages as ORACLE. Note, however, that these pages are known in advance and can be read in an pseudo-optimal way. Algorithms such as the optimal NN-query algorithm by Hjaltason and Samet do not have this advantage and cause many seek operations. This advantage becomes more apparent in our experiments in Section 4. There we will show that the combined overhead of first and second phase is much less than 4 times ORACLE's overhead.

3.2 Quality of r_{second}

The second range query occurs only if r_{first} was too small. Since a theoretical analysis assuming uniformity leads always to an overestimation of r_{first} (implying no need for a second phase), we will only give some experimental findings in this section.

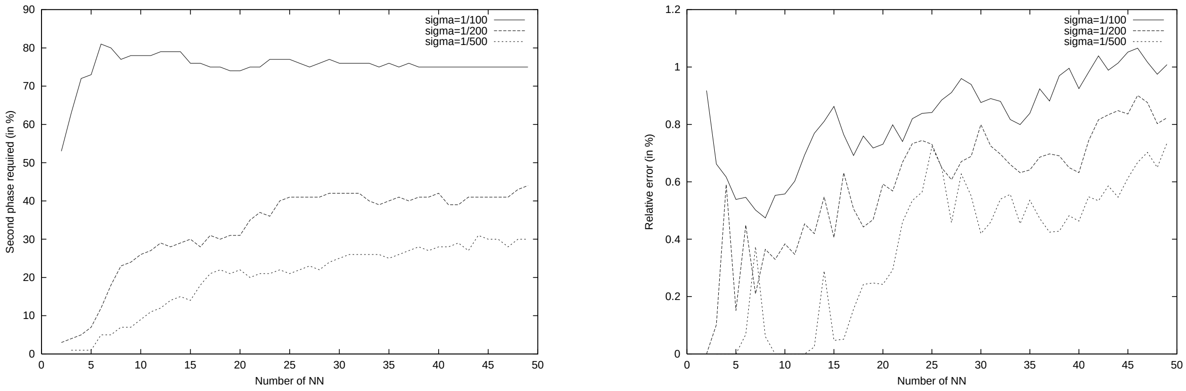
We ran 500 k-NN-queries on the LANDSAT dataset and measured how often a second phase was needed as well as the relative error of r_{second}. Figure 8 shows the results of our experiments for different sample rates. Subfigure 8(a) shows that the probability of a second phase occurrence increases with increasing k for small samples. Note that even though in nearly 80% of the cases a second range query was necessary for $\sigma = 1/100$, the overall cost is low as we will see in Section 3.3.

This can be explained when looking at Subfigure 8(b) which gives the relative error for r_{second}. With increasing k, the error increases slightly again but it stays overall below 1.2% and is therefore negligible. This shows that the points from the pages read in the first phase produce a very accurate second radius estimate.

It has to be noted, however, that this holds only for index structures that cluster points in pages, such as the R*-tree. Non-clustering index structures cause a high overestimation of r_{second}, as we will see for the VA-file in Section 4.

3.3 Combined Overhead of First and Second Phase

In this section, we examine the combined effect of the first and second phase, and the impact of errors in r_{first} on the overall performance. We performed 500 k-NN-queries (with k ranging



(a) Occurrences of second phase (LANDSAT)

(b) Relative Error of r_{second} (LANDSAT)

Figure 8: Analysis of second phase

between 10 and 50) on the LANDSAT dataset. However, instead of using our formula to estimate the first radius, we took the exact NN-radius and perturbed it by some error between -50% and 50%. The result is plotted in Figure 9. All curves are normalized to the optimal number of page

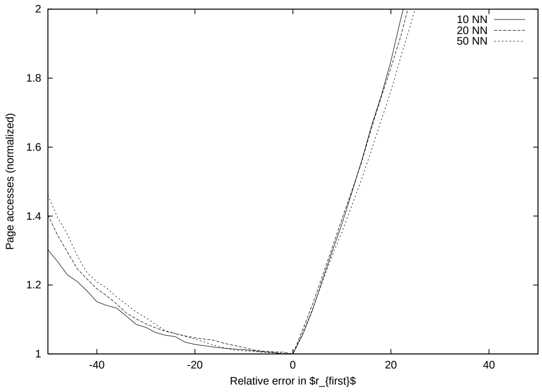


Figure 9: Combined I/O overhead (LANDSAT)

accesses.

As expected, the curve is u-shaped. The more the first radius is underestimated, the more pages have to be read in the second phase (due to a worse r_{second}-estimation). The more the first radius is overestimated, the more pages are read in the first phase. As we can see, underestimating the radius leads to a lower overall cost than overestimating. This can explained by the high quality of r_{second}, even when only a few pages are accessed during the first phase. On the other hand, an overestimation causes always unnecessary pages to be read.

When underestimating the first radius by up to 50% (half the radius size!), only about 40% more pages have to be read compared to ORACLE. However, when overestimating the radius by 20%, more than 80% additional pages have to be fetched. This seems to be a lot but it should be noted that the index consists of about 9,000 pages. Less than 1% of whom are typically fetched during the NN-query. So, even if the number of pages doubles due to the estimation error, still less than

2% of the pages will be read. Additionally, the page reads are performed with a pseudo-optimal read schedule, reducing the cost further.

The next section presents the overall I/O cost considering disk seek and transfer times and shows that the overhead is very low. We also compare our technique with ORACLE and conclude that our technique comes very close to the optimal algorithm.

4 Experimental Results

In order to evaluate our accelerated k-NN-query algorithm, we performed experiments for a large number of datasets. Here we present the results for a synthetic (UNIFORM) and two real datasets (LANDSAT and TEXTURE). For each dataset we ran 100 k-NN queries where k varied between 10 and 50. All queries are density-biased, i.e. query points are picked with higher probability from a region with higher density. The sampling rate was always 1/100. For the underlying harddisk we assume a 20 MB/s transfer rate and an average seek time of 10 ms. The index page capacity is 8 KBytes.

Before discussing the experiments, let us elaborate a bit more on the instantiation of the different components of our framework. The NN-radius estimators are instantiated as discussed in Subsections 2.1 and 2.2. In the following, we will therefore only elaborate on the other components, the disk read schedulers and the index structure for range queries.

4.1 Instantiating the Disk Read Schedulers

The main purpose of the disk read schedulers is to reduce the amount of seeks necessary during range queries. This can be achieved by reading sets of pages in a bulk: once r_{first} is computed, we collect the IDs of the pages to read from disk (i.e. the pages that intersect the query range). After this step, we know the required pages and the gap pages. The gap pages are pages that fall between required pages but do not contain any useful information. In order to minimize the I/O cost, it can be advantageous to read gap pages instead of seeking over them since disk seeks are expensive. Seeger et al. [17] give a heuristic to determine the maximum number g_{max} of gap pages that should be read. If the number of gap pages is more than g_{max}, a seek operation is performed.

In more detail, g_{max} is calculated as

$$g_{max} = \frac{t_{seek}}{t_{xfer}} - \frac{N}{C_{eff} \cdot |P|} - \frac{1}{\log(1 - \frac{C_{eff} \cdot |P|}{N})}$$

where P is again the set of pages to be read in the first range query. The average disk seek and transfer times are denoted by t_{seek} and t_{xfer}, respectively.

The page read scheduler of the second range query is instantiated in a similar way. However, pages read during the first phase are not read anymore. For all the remaining pages we compute the near-optimal read schedule the same way as for the first range query. Typically, only one additional page needs to be read in the second phase.

4.2 Instantiating the Index Structure

For our experiments, we will instantiate the range query index structure with three different indexing schemes: the R*-tree, the X-tree, and the VA-file. In order to show the versatility of our framework, we will also discuss the SS/SR-tree and the Pyramid technique without giving experimental results.

R^*-tree Range queries in the R^*-tree [2] are performed by descending those subtrees whose bounding box overlaps the query range until leaf pages are reached. For each reached leaf page, the entries that fall into the range are determined and returned as result set. Since the whole leaf page has to be read anyway, we can additionally inspect the other points in the page to support the second radius estimator.

X-tree Range queries in the X-tree [7] work the same way as in the R^*-tree. The only difference is the way the bounding boxes are constructed. In our experiments, we used the bulkloaded X-tree as presented in [6]. This bulkloading algorithm ensures that no overlaps occur between the bounding boxes and that the page utilization is maximized.

VA-file For the VA-file [19], we used the following range query algorithm: in a first step, the approximative vectors that intersect the query range are determined. This is done without disk I/O since the approximative vectors are stored in memory. In a second step, the corresponding exact vectors are read from disk. For this step, the disk page scheduler is used to minimize seek time. Since the VA-file organizes the exact vectors in pages, we can inspect the other vectors in a read page to support the second radius estimator.

Note, however, that the VA-file does not cluster points in pages. It rather stores the points in whatever order they are given. This causes a deterioration of the second radius estimation quality as we will see in the experimental section.

SS/SR-tree Both the SS-tree [20] and the SR-tree [12] have range queries similar to the R^*-tree. The only difference is the bounding shape. Our technique will therefore yield results similar to the R^*-tree.

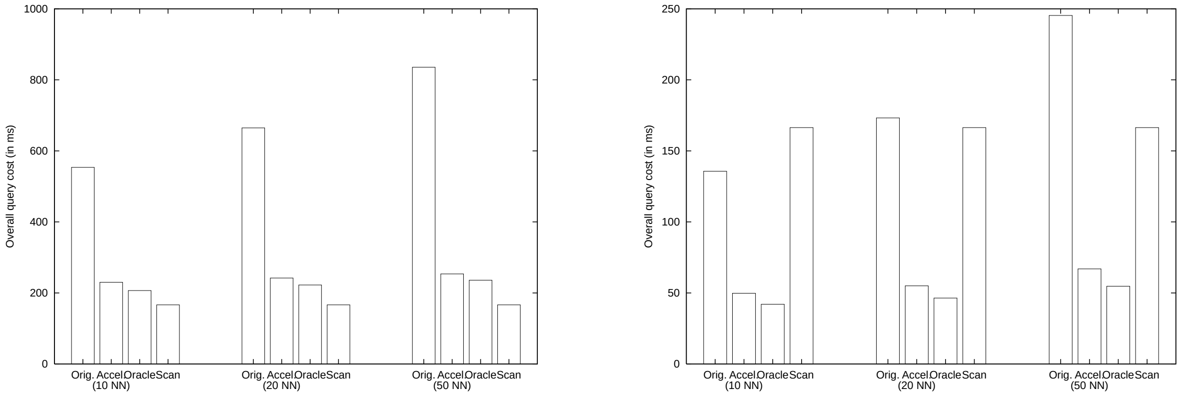
Pyramid Technique The Pyramid technique [5] partitions the dataspace into $2d$ hyper-pyramids. These pyramids are then used to map the d-dimensional points onto 1-dimensional values that can be stored in a B+-tree. In [5], Berchtold et al. give a range query algorithm for this index structure. They transform the d-dimensional query range into a set of 1-dimensional range queries for the B+-tree. During the execution of such a range query, B+-tree pages need to be fetched from disk. The disk read scheduler can be used to accelerate this step. Additionally, the points that are stored in a page but not included in the range query, can be used to support the second radius estimator. Due to the design of the pyramid mapping function, some amount of locality between points is preserved. We can therefore expect meaningful second radius estimates.

However, the main achievement is the support of NN queries for an index structure that offers only range queries.

4.3 Results for Synthetic Data

In this and the next subsection, we compare for three index structures the performance of the original NN-query algorithm with our accelerated version, the optimal algorithm ORACLE, and the linear scan. Recall that ORACLE always picks the correct query radius for the first range query and reads therefore always the minimal number of pages possible for our algorithm. The three index structures we examine are the R^*-tree, an index structure with the bulkloading scheme developed in [6] (we will refer to this structure with "X-tree" since it results in the same page layout as the bulkloaded X-tree), and the VA-file. In case of the R^*-tree and the X-tree, we used the optimal NN-query algorithm proposed by Hjaltason and Samet [10]. For the VA-file, we used the near-optimal NN-query algorithm proposed by Weber and Blott [19]. The linear scan reads all pages in a linear fashion and therefore requires no seek operations.

The first set of experiments was performed on the 8-dimensional UNIFORM dataset. Figure 10 shows the results for the R*-tree and X-tree. For each index structure and k-value, we give four I/O



(a) R*-tree

(b) X-tree

Figure 10: Overall query cost for UNIFORM dataset

costs: for the original query algorithm (denoted by "Orig."), for our accelerated query algorithm (denoted by "Accel."), for ORACLE (denoted by "Oracle"), and for the linear scan (denoted by "Scan"). Each cost value is an average over 100 queries.

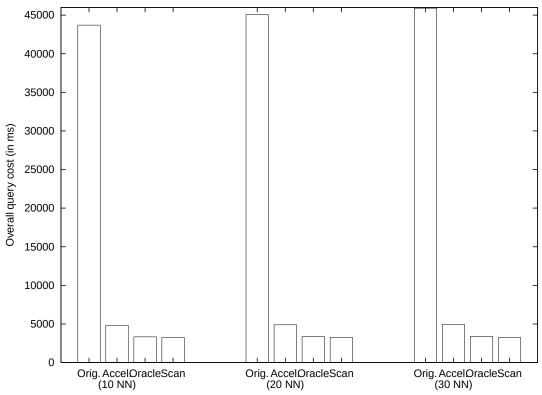
Subfigure 10(a) shows the I/O costs for the R*-tree. The cost of the original query algorithm increases with increasing k and is well above the linear scan. The accelerated query algorithm performs slightly worse than the linear scan. This is due to the high amount of overlaps in the bounding boxes of an R*-tree. The same holds for ORACLE: due to the high overlap, a large amount of pages have to be read, resulting in a higher read cost than for the linear scan[3]. Despite these overlaps, our technique speeds the NN-query up by a factor of $2-4$ (depending on k) and resembles ORACLE closely. Note that our technique can never perform worse than two linear scans over a subset of the data (when employing an optimal page read scheduler), since we perform at most two range queries. Obviously, the original query algorithm cannot give this guarantee.

Subfigure 10(b) gives the results for the X-tree. Since we use an elaborate bulkloading algorithm to build the tree, the page utilization is high and no page overlaps occur. This improves the query performance drastically. For 10-NN queries, the index is even faster than the linear scan. For larger k, the performance deteriorates again and becomes worse than scanning. When using our acceleration technique, the performance is improved even further, as can be seen in the second bars. Compared to the original X-tree, we achieve speed-ups between 3 and 5. Additionally, the accelerated query algorithm is at least 3 times faster than the linear scan.

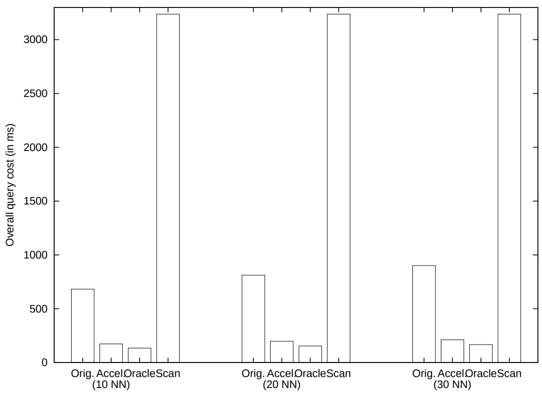
[3] An optimal page read scheduler would have caused the I/O cost of ORACLE to be slightly below the linear scan. However, for performance reasons, we use a page read scheduler that gives only pseudo-optimal schedules, resulting in slightly higher costs. Thanks to the modular design of our framework, other schedulers can be used.

4.4 Results for Real Data

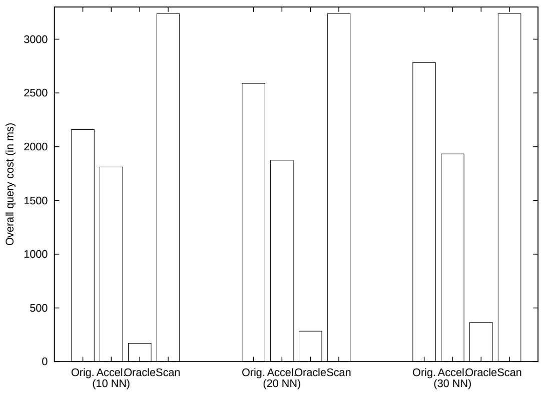
In this section, we give experimental results for two real datasets. The first set of experiments was performed on the LANDSAT dataset. The corresponding plots can be found in Figure 11.



(a) R*-tree



(b) X-tree



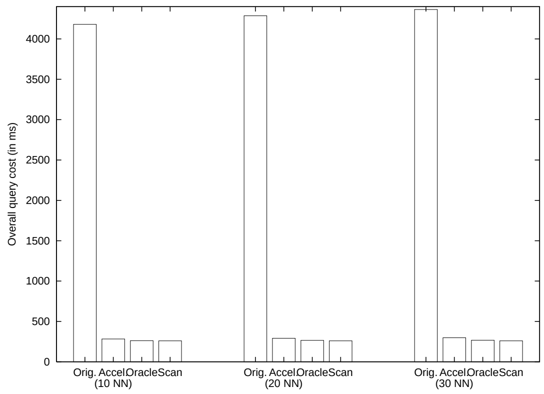
(c) VA-file

Figure 11: Overall query cost for LANDSAT dataset

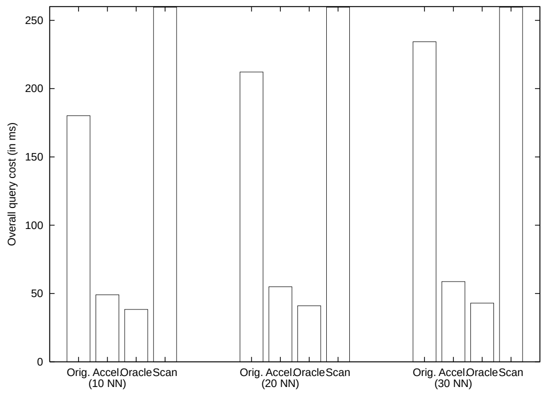
In case of the R*-tree (Subfigure 11(a)), we observe much higher query costs than for the UNIFORM dataset. The reason is the increased amount of overlap that occurs for this 60-dimensional dataset. When applying our technique, the I/O cost is reduced by a factor of about 9. The cost again ranges between 1 and 2 linear scans and only slightly above ORACLE.

Two points should be noted from the X-tree plots (Subfigure 11(b)). First, the X-tree stays well below the linear scan for all queries. This is even more remarkable considering the fact that the X-tree algorithm performs only random page accesses, whereas the linear scan uses only sequential ones. This shows that for highly clustered data, a well tuned index structure is still very useful for NN-queries.

Even more remarkable is the second observation that our accelerated query algorithm outperforms this well-tuned index structure by a factor of $3-5$. The reason is simple. Even though the accelerated query algorithm needs to read more pages, it knows them in advance and can read as many of them sequentially as necessary. In a way, it gets the best of both worlds, the X-tree and



(a) R*-tree



(b) X-tree

Figure 12: Overall query cost for TEXTURE dataset

the linear scan: it reads nearly as few pages as the optimal NN-query algorithm of the X-tree, and it performs nearly as few seeks as the linear scan.

Subfigure 11(c) gives the performance results for the VA-file. Note that the VA-file performance deteriorates similar to the R*-tree and X-tree with increasing k. Moreover, its performance even approaches the linear scan. The reason for this behavior lies in the second phase of the VA-file query algorithm: since the second phase reads the exact vectors according to their approximative distance from the query point, most disk accesses will be random. The linear scan, on the other hand, causes no random seeks.

Even though our algorithm improves the query performance of the VA-file, the speed-up is less than for the other index structures. This can be explained by the way the index structures cluster the data. The R*-tree and X-tree perform a clustering of points in pages, whereas the VA-file does not. It stores the points in whatever order they were given. Distant points can therefore end up in the same VA-file page. Since our algorithm determines the second radius estimate from the points found in the pages of the first range query, this can cause a high overestimation. A large second radius estimate leads to high I/O costs for the second range query. This can also be inferred from the ORACLE performance: since ORACLE knows the exact query radius, its cost is much lower than for our algorithm. We can therefore conclude that our second radius estimator works best for index structures that perform clustering of points in pages, while non-clustering index structures cause a higher I/O cost in the second phase.

Finally, Figure 12 gives the experimental results for the TEXTURE dataset. The result looks similar to the LANDSAT dataset. Our accelerated approach performs about 15 times faster than the R*-tree and $3-4$ times faster than the X-tree. The accelerated X-tree shows a speed-up of 5 compared to the linear scan and resembles ORACLE closely.

5 Previous Work

Many researchers have investigated models for predicting the performance of index structures. Some of this work focuses on predicting the index performance for an average workload, some of it focuses on the expected cost of a single query. Since the technique presented in this paper aims at accelerating single queries, we will concentrate on the latter body of work.

The work that comes most closely to our technique was presented by Berchtold et al. [4]. They propose a new index structure, called IQ-tree, and give a probability-based method to optimize page reads during NN queries. The authors estimate the probability that a page will be read during a query and use this probability to decide which pages should be read together with the next page in order to avoid expensive seeks. The probability that a page will be accessed is defined as the percentage of the page volume covered by the NN sphere.

In contrast to our technique, this model assumes uniform data distribution within the pages. This can lead to high estimation errors in the access probability. Since the page reads are controlled by this probability estimate, many unnecessary pages are accessed. Furthermore, at least one page needs to be accessed to get an initial radius estimate.

Other authors extend index structures by statistical information, which can be used to estimate the query radius and the query cost. Ciaccia et al. [9] extend the M-tree nodes by statistics (the distance distribution) in order to predict the CPU and I/O cost of range and NN queries. Most work is based on histograms over the dataset. Theodoridis and Sellis [18] give a model for predicting the performance of range queries on R*-trees. They generate what they call a *density surface* which is basically a two-dimensional histogram representing the local densities of the dataset. Acharya et al. [1] show how the prediction error of histograms can be reduced by reducing the variance of densities within the histogram regions. Korn et al. [13] demonstrate how discontinuities at the histogram region edges can be avoided by using splines. Finally, Jin et al. [11] extend the histogram approach for spatial non-point data.

The advantage of these techniques is their high accuracy in modeling data by considering local effects. A disadvantage of the histogram approaches is that they are not applicable in high dimensions since either the number of histogram regions becomes too large or these regions contain too much empty space and become inaccurate.

In [16], sampling is used to overcome this problem. In contrast to this paper, the sample is used to predict the overall query cost of a given index structure. More specifically, the sample is used to predict the index page layout. Here, on the other hand, we use the sample to predict the query radius for one particular query. Moreover, we employ the fractal dimensionality to compensate for sampling, while in [16], uniformity is assumed to adjust the page layout prediction.

We are not aware of any framework that transforms NN queries into at most two range queries and is thereby able to speed up the query performance of any given index structure.

6 Conclusions

We showed in this paper that it is possible to accelerate the performance of high-dimensional NN-queries by transforming the NN-query into at most 2 range queries. As one major contribution, we presented a general framework to achieve this. The framework can be instantiated to a large group of index structures, radius estimators, and disk read schedulers. As a by-product, we showed how the algorithm can be modified to support approximate NN-queries.

We discussed one possible radius estimator for the first and the second phase of our framework. Both estimators are novel and establish the second major contribution of this paper. The first estimator uses a data sample and the fractal dimensionality of the data. Our extensive analysis for uniform data indicated that the estimation error is below 14% and that errors in the fractal dimensionality estimation have only minor impact on the overall cost. The second estimator takes points from already read pages into account. It works therefore best if the underlying index structure clusters points in pages.

Our analysis for real data demonstrated that the first radius estimate is already very accurate. In many cases, no further range queries are necessary. If a second phase does occur, the first radius estimate was already very close to the correct one and only a few (typically only one) additional pages have to be fetched. If no second phase occurs, the number of pages to be read rarely exceeds twice the optimal number of pages. As a hard bound, we can guarantee that our algorithm never performs worse than 2 linear scans over a subset of the data. This analysis represents the third major contribution of this paper.

In the experimental section, we showed that our technique performs up to 15 times faster than the R*-tree and nearly 5 times faster than a bulkloaded X-tree-like index structure. Even the VA-file, which does not cluster points in pages, benefits from our framework. Surprisingly, our technique outperformed the NN-query algorithms proposed as part of these index structures. Additionally, we discussed how the SS/SR-tree can be accelerated by our technique and how the Pyramid technique can be easily extended by NN-queries.

For the future, we plan on extending our technique to index structures for general metric spaces (e.g. M-tree) and non-metric spaces (e.g. Suffix-trees). Second, we will examine the applicability of our technique to other query types, such as spatial joins. Finally, since dimensionality reduction techniques (such as KLT) are widely used for high-dimensional data, it would be interesting to make our technique available to index structures storing only a subset of the data dimensions.

References

[1] Swarup Acharya, Viswanath Poosala, and Sridhar Ramaswamy. Selectivity estimation in spatial databases. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 13–24, 1999.

[2] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 322–331, 1990.

[3] Alberto Belussi and Christos Faloutsos. Estimating the selectivity of spatial queries using the 'correlation' fractal dimension. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 299–310, 1995.

[4] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander. Independent quantization: An index compression technique for high-dimensional data spaces. In *Proc. Int. Conf. on Data Engineering*, 2000.

[5] S. Berchtold, C. Böhm, 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.

[6] Stefan Berchtold, Christian Böhm, and Hans-Peter Kriegel. Improving the query performance of high-dimensional index structures by bulk load operations. In *Proc. Int. Conf. on Extending Database Technology*, pages 216–230, 1998.

[7] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. The X-tree : An index structure for high-dimensional data. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 28–39, 1996.

[8] Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proc. Int. Conf. Database Theory*, pages 217–235, 1999.

[9] P. Ciaccia and M. Patella. Bulk loading the M-tree. In *9th Australasian Database Conference (ADC'98)*, pages 15–26, 1998.

[10] Gisli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *Advances in Spatial Databases — Fourth International Symposium (SSD)*, pages 83–95, 1995.

[11] J. Jin, N. An, and A. Sivasubramaniam. Analyzing range queries on spatial data. In *Proc. Int. Conf. on Data Engineering*, 2000.

[12] Norio Katayama and Shin'ichi Satoh. The SR-tree: An index structure for high-dimensional nearest neighbor queries. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 369–380, 1997.

[13] Flip Korn, Theodore Johnson, and H. V. Jagadish. Range selectivity estimation for continuous attributes. In *Proc. International Conference on Scientific and Statistical Database Management*, pages 244–253, 1999.

[14] Flip Korn, Bernd-Uwe Pagel, and Christos Faloutsos. Deflating the dimensionality curse using multiple fractal dimensions. In *Proc. Int. Conf. on Data Engineering*, 2000.

[15] Christian A. Lang, Hakan Ferhatosmanoglu, Divyakant Agrawal, Amr El Abbadi, and Ambuj K. Singh. Inherent dimensionality of high-dimensional data. Unpublished manuscript, University of California, Santa Barbara.

[16] Christian A. Lang and Ambuj K. Singh. Modeling high-dimensional index structures using sampling. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2001.

[17] Bernhard Seeger, Per-Åke Larson, and Ron McFayden. Reading a set of disk pages. In *Proceedings of the Int. Conf. on Very Large Data Bases*, pages 592–603, 1993.

[18] Yannis Theodoridis and Timos K. Sellis. A model for the prediction of R-tree performance. In *Proc. ACM Symp. on Principles of Database Systems*, pages 161–171, 1996.

[19] R. Weber and S. Blott. An approximation based data structure for similarity search. Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997.

[20] David A. White and Ramesh Jain. Similarity indexing with the SS-tree. In *Proc. Int. Conf. on Data Engineering*, pages 516–523, 1996.