International Journal of Image and Graphics
Vol. 3, No. 1 (2003) 3–29
© World Scientific Publishing Company

FASTER SIMILARITY SEARCH FOR MULTIMEDIA DATA VIA QUERY TRANSFORMATIONS

CHRISTIAN A. LANG[*] and AMBUJ K. SINGH[†]
Department of Computer Science, University of California, Santa Barbara
Santa Barbara, CA 93106, USA
[*]*clang@cs.ucsb.edu*
[†]*ambuj@cs.ucsb.edu*

Received 7 August 2002
Revised 26 September 2002

The performance of nearest neighbor (NN) queries degrades noticeably with increasing dimensionality of the data due to reduced selectivity of high-dimensional data and an increased number of seek operations during NN-query execution. If the NN-radii would be known in advance, the disk accesses could be reordered such that seek operations are minimized. We therefore propose a new way of estimating the NN-radius based on the fractal dimensionality and sampling. It is applicable to any page-based index structure. We show that the estimation error is considerably lower than for previous approaches. In the second part of the paper, we present two applications of this technique. We show how the radius estimations can be used to transform k-NN queries into at most two range queries, and how it can be used to reduce the number of page reads during all-NN queries. In both cases, we observe significant speedups over traditional techniques for synthetic and real-world data.

Keywords: Multimedia data; high-dimensional indexing; query transformation; similarity search.

1. Introduction

Nearest neighbor (NN) queries are an important query type for high-dimensional data, such as multimedia data, strings, and time sequences. One example for such a query is “Get the 10 closest images to a given query image” where similarity is typically defined on color distributions or texture features. Another example is DNA sequencing. Here, for each gene segment of one genome, all closest matching gene segments of another genome have to be found. Instead of asking for one best match, this query type asks for all best matches.[a]

On large databases, such queries can be very time-consuming, especially when the number of feature dimensions is high. The main reason for the typically high

[a]Analogously, k-NN queries ask for the k best matches.

response times lies in the fact that index structures supporting these queries are disk-based and therefore require a large amount of disk I/Os for answering queries.

Much research has therefore focused on accelerating the execution of such queries by reducing the number of disk pages to read, or by reducing the number of disk seek operations during page reads. Techniques that try to reduce the number of seeks cannot guarantee a minimum number of pages to be read, and vice versa. One extreme example is the linear scan for answering NN queries: by scanning the whole data file, the number of seek operations is minimized but at the same time, all pages have to be accessed. An example for the other extreme is Hjaltason and Samet's k-NN query algorithm.[1] Their algorithm accesses only a minimal number of disk pages but induces a high amount of seek operations.

An algorithm that obtains the best of both worlds would be reading only the pages that contain the NNs with a read schedule that minimizes seeks. In other words, the NN query is transformed into a range query. The problem with this approach is that we need to know the required pages in advance. In order to determine these pages, it would be helpful to know the range of the query containing the NNs. Much work in the last years went into predicting such query ranges. However, current prediction techniques are not sufficient for this task since they predict an *average* query range for the entire dataspace rather than for a single query.

A major contribution of this work is therefore an improved estimate of the query range based on the fractal dimensionality of data and sampling. We distinguish between two types of estimates: query-independent ones and query-dependent ones. For the former, we introduce two improvements on the fractal dimensionality-based estimation of the average query range. The latter type assumes that the query is known and a range estimate is desired. We show how the new query-independent estimate can be used to obtain an (even tighter) query-dependent query range estimate through sampling. The basic idea is to get a first estimate on the sample and then use the query-independent estimates to compensate for the effects due to sampling.

As a second contribution, we introduce a way of obtaining a close upper bound on the NN range for page-based index structures. This bound is obtained by inspecting all points stored in pages that are intersected by the query-dependent range estimate. The NNs found in these pages are then used to obtain an upper bound on the query range estimate. Our experiments show that this estimate is within 1% of the actual radius.

As a third contribution, we introduce two new NN query algorithms based on these estimates. The first algorithm is for single k-NN queries, and the second algorithm is for all-NN queries. Both new algorithms reduce the amount of disk seeks and page reads at the same time by using our new query-dependent range estimates to determine which pages will have to be fetched from disk in the future. This set of pages is then read into memory using a read schedule that minimizes disk seeks. In case the query-dependent range estimator underestimates the real radius (and therefore would miss some NNs), we will use our second estimation technique to obtain an upper bound on the query range. For both query types, we observed a significant reduction in overall query time.

The paper is organized as follows. Section 2 discusses related work. In Sec. 3, we show how the fractal dimensionality-based estimation of the average query range can be improved. This is then used together with sampling to obtain a query-dependent way of calculating a query range estimate in Sec. 4. Section 5 analyzes the quality of the obtained estimates. Section 6 introduces a technique to compute a tight upper bound on the query-dependent query range. Sections 7 and 8 present two applications of these estimates and we conclude in Sec. 9.

2. Previous Work

2.1. *Fractal dimensionality and NN-radius estimation*

Faloutsos *et al.*[2] presented the first cost model for R-trees based on the fractal dimensionality which is claimed to be the "inherent dimensionality" of a dataset. This first model was restricted to range queries but later works by Papadopoulos and Manolopoulos[3] extended it for 1-NN queries in R-trees. Korn *et al.*[4] present a version for k-NN queries which we discuss briefly in the following.

Korn *et al.*[4] show how two different fractal dimensionality measures can be used to describe certain properties of datasets. One of these measures is the so-called *correlation fractal dimensionality* D_2. It is defined as follows[b]:

Definition 1 (Correlation Fractal Dim). *For a point set that has the self-similarity property in the range of scales* $r \in (r_1, r_2)$, *its correlation fractal dimensionality* D_2 *for this range is measured as*

$$D_2 \equiv \frac{\partial \log \sum_i p_i^2}{\partial \log(r)} = \text{const.} \quad r \in (r_1, r_2)\,,$$

where p_i *is the percentage of points which fall inside the* i*th cell when dividing the data space into hypercubic grid cells of side* r.

Korn *et al.*[4] show how this measure can be used to estimate the k-NN query radius for arbitrary datasets. They show that for the L_∞-norm, the average k-NN radius is given by

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

where N is the number of data points. Section 3.1 shows how D_2 can be computed for a given dataset using a box-counting algorithm and how this computation can be improved.

[b]A summary of all symbols used in this paper can be found in Table 1.

Table 1. Notation used in the paper.

N	number of data points
σ	data sampling rate
E	embedding dimensionality (i.e. number of attributes)
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_{expected}	query-dependent k-NN radius estimate
r_{upper}	upper bound on k-NN radius
$d_{nn}(k)$	radius estimate based on D_2 (full dataset)
$d_{\text{sample}-nn}(k)$	radius estimate based on D_2 (sample data)
$d_{nn}^{\text{cutoff}}(k)$	radius estimate with boundary effects (full dataset)
$d_{\text{sample}-nn}^{\text{cutoff}}(k)$	radius estimate with boundary effects (sample data)
k	number of NNs

2.2. *Accelerated k-NN query algorithms*

Many researchers have investigated models for predicting the performance of index structures for query optimization. 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 Chen and Ling.[5] They show how sampling can be used to predict the k-NN radius and accelerate NN queries. However, their technique does not make use of the fractal dimensionality of the data in order to compensate for sampling errors. Instead, they require a calibration phase consisting of sample queries. This might introduce high errors unless many sample queries are performed.

Berchtold *et al.*[6] propose a new index structure, called the 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.*[7] 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[8] 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.*[9] show how the prediction error of histograms can be reduced by reducing the variance of densities within the histogram regions. Korn *et al.*[10] 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.

Lang and Singh[12] use sampling to overcome this problem. In contrast to this paper, there 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 Lang and Singh[12] assume uniformity to adjust the page layout prediction.

2.3. *Accelerated All-NN query algorithms*

Several approaches[13,14] focus on efficient processing of spatial joins on R-trees. However, they do not take seek and transfer costs into account. Instead, they try to minimize the number of page misses during a join. Since these algorithms perform the join by navigating through the index tree rather than through the disk pages, they cause many random page accesses.

Other work[15,16] investigates new indexing and query processing techniques in order to reduce the number of page accesses during spatial joins. In contrast to our approach, these techniques do not make use of existing index structures and are restricted to join conditions of the type $\sigma(x, y) =$ "the distance between x and y is less than d".

Hjaltason and Samet[17] propose an incremental join algorithm that returns the closest join pair first. This work does not try to accelerate the search for the full join result as we do.

A different approach by Braunmüller *et al.*[18] reduces the I/O and CPU-cost of multiple NN-queries. They process one query after the other but take results of older queries into account in order to accelerate newer ones. This process is certainly faster than single NN-queries but it can be expected to perform worse when searching for all NNs since the page accesses are still random. Our approach, on the other hand, tries to maximize the number of sequential page accesses.

3. Query-Independent *k*-NN Radius Estimation

In this section, we show how the NN radius estimation based on the fractal dimensionality, as presented by Korn *et al.*,[4] can be improved in two ways. First, we improve the computation of D_2 by using smaller steps for the box-counting algorithm. And second, we show how boundary effects can be taken into account when estimating the radius.

3.1. *Improved fractal dimensionality estimation*

The correlation dimension D_2 of point datasets is typically computed using a box-counting algorithm,[19] as shown in Fig. 1.[c] With each iteration of the algorithm, another point on the $r/N(r)$-graph is computed. A typical graph is shown in Fig. 2. It roughly consists of two constant tails, one at log(1) (on the right) and one at $\log(N)$ (on the left), and a linear piece inbetween. Towards the right end, r becomes large enough to cover the whole dataset at some point, leading to the log(1)-tail. Towards the left end, r becomes so small that every data point falls into its own

```
Input:   normalized dataset
Output:  fractal dimensionality

(1) Let $r := 1$.
(2) Partition the data space in $\left(\frac{1}{r}\right)^d$ boxes of side length $r$
    and count the number of boxes $N(r)$ containing points.
(3) Let $r := \frac{r}{2}$. If $r$ is not small enough,[d] go to (2).
(4) Find the linear part of the $r/N(r)$-graph
    in log/log-scale and compute its slope, return it.
```

Fig. 1. The box counting algorithm.

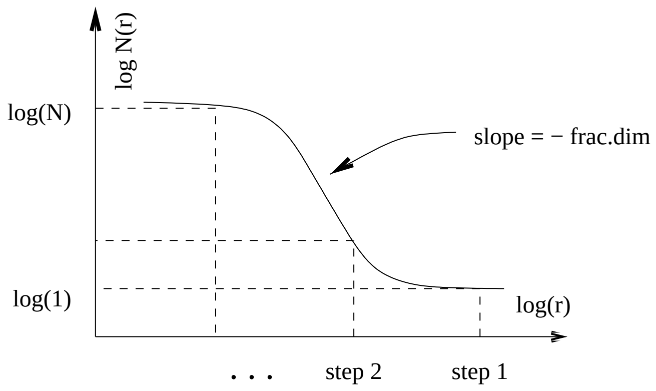
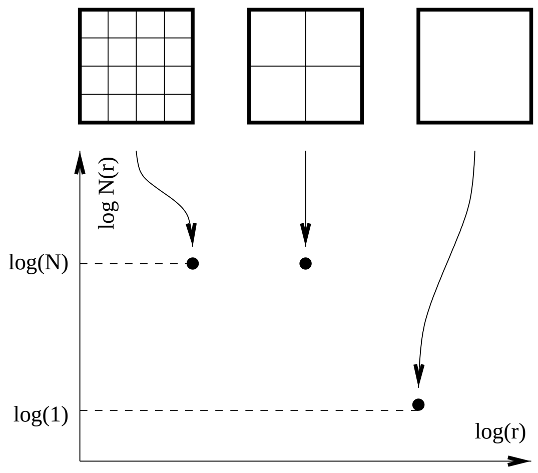
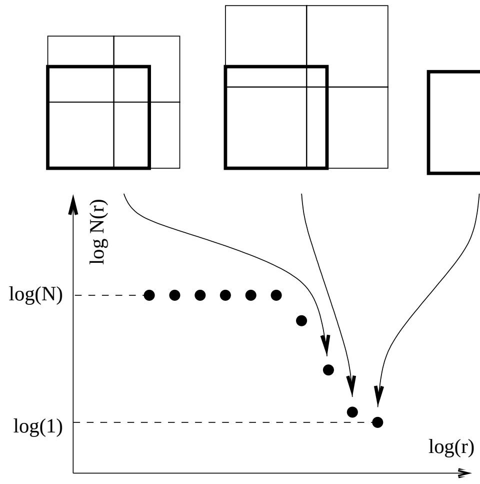


Fig. 2. Typical $r/N(r)$-plot.

[c]Note that the presented algorithm is a slightly simplified version of the algorithm for computing D_2. The basic idea is the same, however.



(a) Original box counting algorithm



(b) Modified box counting algorithm

Fig. 3. Relation between box coverage and log/log-plot.

box, causing the number of occupied boxes to reach N. For fractal datasets, a linear section exists inbetween the tails. Its slope determines the fractal dimensionality.

The box counting algorithm partitions the data space into $b(i) := 2^{\mathrm{d}\cdot(i-1)}$ boxes in the ith iteration. For uniform data, there are $\binom{b(i)+N-1}{N}$ possible distributions of N data points in the $b(i)$ boxes. It follows easily that the probability that a box contains no points is $(b(i)-1)/(b(i)-1+N)$. For $i = 2$, $N = 10^6$ and $d = 50$, this

[d]By "small enough," we mean small enough for the linear part of the graph to be revealed.

```
Input:    normalized dataset
Output:   fractal dimensionality

(1) Let $r := 1$.
(2) Partition the data space in $\lceil \frac{1}{r} \rceil^d$ boxes of side length $r$
    and count the number of boxes $N(r)$ containing points.
(3) Let $r := \frac{r}{r_{\text{div}}}$. If $r$ is not small enough, go to (2).
(4) Find the linear part of the $r/N(r)$-graph
    in log/log-scale and compute its slope, return it.
```

Fig. 4. The modified box counting algorithm.

probability is nearly 1. This means that nearly all boxes are empty. It follows that with high probability the number of boxes with points is N after the first iteration. In these cases, no linear slope can be found in Step (4) of the algorithm. This is shown in Fig. 3(a).

Even though this problem is less severe for clustered data, in most cases the number of points on the linear slope is still too small for accurate calculations. For that reason, we modify the box counting algorithm as follows: instead of dividing r by 2 in Step (3), we use a value $1 < r_{\text{div}} < 2$. The boxes in direction of higher coordinates may now partly fall outside the data space [cf. Fig. 3(b)] but since they still form a complete coverage of the points, the computed dimension is correct. The modified algorithm is given in Fig. 4. Now, more points can be computed between the maximal ($\log N$) and minimal ($\log 1$) values of the graph.

3.2. *Taking boundary effects into account*

The k-NN radius estimate can be further improved by taking boundary effects into account. The formula for $d_{nn}(k)$ is derived by assuming that queries that exceed the limits of the data space are wrapped around. In other words, in a data space with extents normalized to 1, a query with radius $\epsilon = 1/2$ is assumed to cover the whole data space. In reality however, query shapes are cut off at the data space boundary. In order to cover the same number of NNs, the NN radius has to increase correspondingly. This effect becomes especially pronounced in high dimensions.

The expected volume of a query hypercube in E dimensions with radius r is

$$(2r)^E$$

if no boundary effects are taken into account.

With boundary effects, this volume depends on the location of the query point. The expected volume $\widetilde{\text{vol}}(r)$ is then

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

or

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

or, integrated,

$$\left(\frac{1}{2} + r + \frac{1}{2}\left(\min\{1-r, r\}^2 + \max\{1-r, r\}^2\right) + r \min\{1-r, r\} \right.$$
$$\left. - (1+r)\max\{1-r, r\} \begin{cases} 2r - 4r^2 & \text{if } r \le \dfrac{1}{2} \\ 2r - 1 & \text{otherwise} \end{cases} \right)^E .$$

Then the expected k-NN radius is the r for which

$$k = (N-1) \cdot \widetilde{\mathrm{vol}}(r)^{\frac{D_2}{E}} . \tag{2}$$

We will denote this r by $d_{nn}^{\mathrm{cutoff}}(k)$ in the following.

In order to evaluate the quality of the new query-independent radius estimates, we computed the box counting dimension for a large number of high-dimensional real-world datasets. We then used Eq. 1 to estimate the k-NN radius. In all cases, the values returned by the modified algorithm gave better radius estimations than the original algorithm. Table 2 shows how the relative error in NN-radius estimation changes for different values of r_{div} and k for the LANDSAT-dataset.[e] The error was calculated by comparing the radius estimate $d_{nn}^{\mathrm{cutoff}}(k)$ with an average NN-radius obtained by running 500 random k-NN queries. As can be seen, by using finer step sizes in the box counting algorithm, the estimation error can be reduced significantly. As apparent, the obtained value for D_2 is more accurate when more intermediate steps are computed. Note, however, that this value will remain constant from a certain step size on. This is due to the discrete nature of the underlying dataset. The optimal value of r_{div} therefore depends on the dataset and remains the subject of future research.

Table 2. NN-radius estimation errors.

r_{div}	Rel. error (10 NN)	Rel. error (20 NN)	Rel. error (50 NN)
2 (Orig. alg.)	49.29%	58.52%	84.35%
1.15 (5 $\times$ more steps)	−19.91%	−11.43%	8.66%

[e]This dataset consists of 275 000 60-dimensional texture feature vectors extracted from satellite images.

4. Query-dependent k-NN Radius Estimation

The box counting algorithm allows the estimation of the average k-NN radius. This calculation can be done statically for a given dataset. It is possible to achieve better results if we know the actual query location and have a data sample available. We will refer to this case as *query-dependent radius estimation.*

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*.[f] 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.

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. We can use the query-independent radius estimate $d_{nn}^{\text{cutoff}}(k)$ for this purpose. Similar to $d_{nn}^{\text{cutoff}}(k)$, we can calculate the k-NN radius $d_{\text{sample}-nn}^{\text{cutoff}}(k)$ for the sample dataset. Note that both values can be precomputed for a given dataset and sample.

Once $d_{nn}^{\text{cutoff}}(k)$ and $d_{\text{sample}-nn}^{\text{cutoff}}(k)$ are known, we also know the expected rate of change in the NN-radius when moving from the sample to the full dataset, namely $d_{nn}^{\text{cutoff}}(k)/d_{\text{sample}-nn}^{\text{cutoff}}(k)$. The query dependent radius estimate r_{expected} is therefore

$$r_{\text{expected}} = \frac{d_{nn}^{\text{cutoff}}(k)}{d_{\text{sample}-nn}^{\text{cutoff}}(k)} \cdot r_{\text{sample}} \, . \tag{3}$$

The quality of this query-dependent estimate is discussed in the next section.

5. Quality of the Query-dependent Radius Estimate

We first examine analytically the deviation of the radius estimate from the correct NN-radius for uniform datasets (Sec. 5.1). We then show that similar numbers hold for real datasets (Sec. 5.2).

5.1. *Expected error for uniform data*

In order to derive a formula for the expected error of r_{expected}, let us assume a normalized dataspace. According to Belussi and Faloutsos,[19] the average number of neighbors k of a point within a region of regular shape and radius $\tilde{r}$ is then given by

$$k = (N-1) \cdot \text{vol}(\tilde{r})^{\frac{D_2}{E}} \, ,$$

where $\text{vol}(\tilde{r})$ denotes the volume of the region with radius $\tilde{r}$. One the other hand, we can use this formula to compute the expected query radius $\tilde{r}$ via some root

[f]We assume the sample is obtained by randomly selecting $N \cdot \sigma$ points of the dataset.

finding method. In that case, the value $\mathrm{vol}(\tilde{r})$ has to denote the volume of the query shape with radius $\tilde{r}$ after being cut off at the dataspace boundary. Details on the computation of this volume can be found in Sec. 3.2.

Since $D_2 = E$ for uniform data, the above equation is equivalent to

$$k = (N - 1) \cdot \mathrm{vol}(\tilde{r})\,.$$

The same holds for the sample, resulting in the following equation for the expected sample radius $\widetilde{r_{\text{sample}}}$:

$$k = (N \cdot \sigma - 1) \cdot \mathrm{vol}(\widetilde{r_{\text{sample}}})\,.$$

Thus, we know the expected correct query radius for the full dataset and the sample. However, what is the expected value of r_{expected}? Since we assume uniformity, $D_2 = D_2'$ and therefore[g]

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

Therefore,

$$r_{\text{expected}} \approx \widetilde{r_{\text{sample}}} \cdot \sigma^{\frac{1}{D_2}}\,. \tag{4}$$

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

$$\frac{r_{\text{expected}} - \tilde{r}}{\tilde{r}}\,,$$

which is plotted in Fig. 5 for varying k ($N = 100\,000, E = 60$). 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 query-dependent 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_{expected} overestimates the correct radius. Real data, however, can cause an underestimation as we will see in the next section.

5.2. *Measured error for real data*

This section shows how the relative error of r_{expected} varies for the LANDSAT dataset which contains more than a quarter million 60-dimensional points and is highly clustered. Figure 6 shows that — similar to the uniform case — with increasing k, the relative error of r_{expected} 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 of the last section. First, for a sample rate of 1/100, the relative error drops below zero for $k > 3$. This means that r_{expected} underestimates the real radius.

[g]For simplicity, we use d_{nn} here rather than d_{nn}^{cutoff}. Note that the real expected error is therefore lower than this calculation suggests.

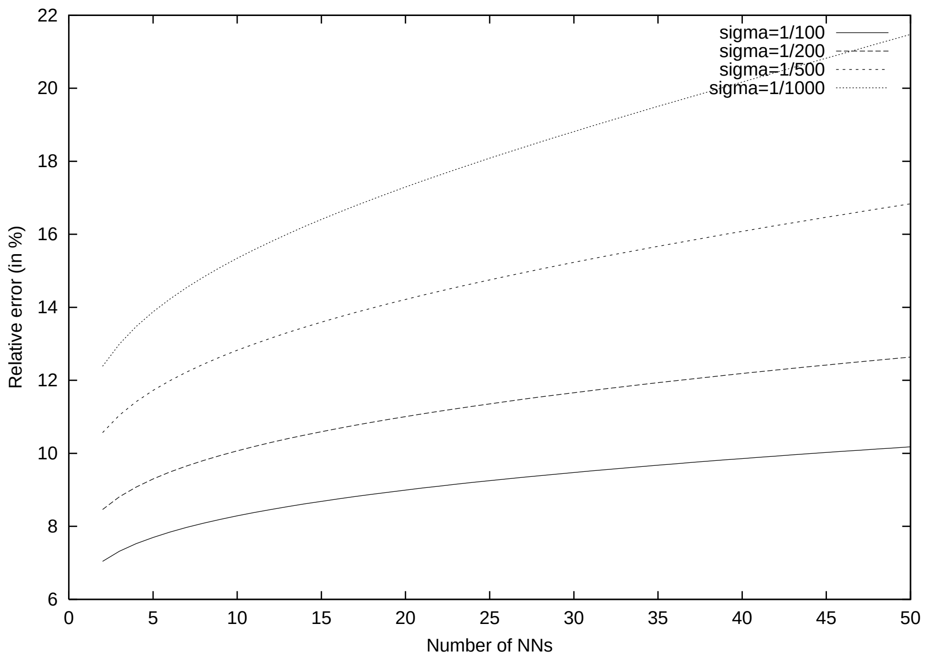


Fig. 5. Rel. error of r_{expected} (analytical).

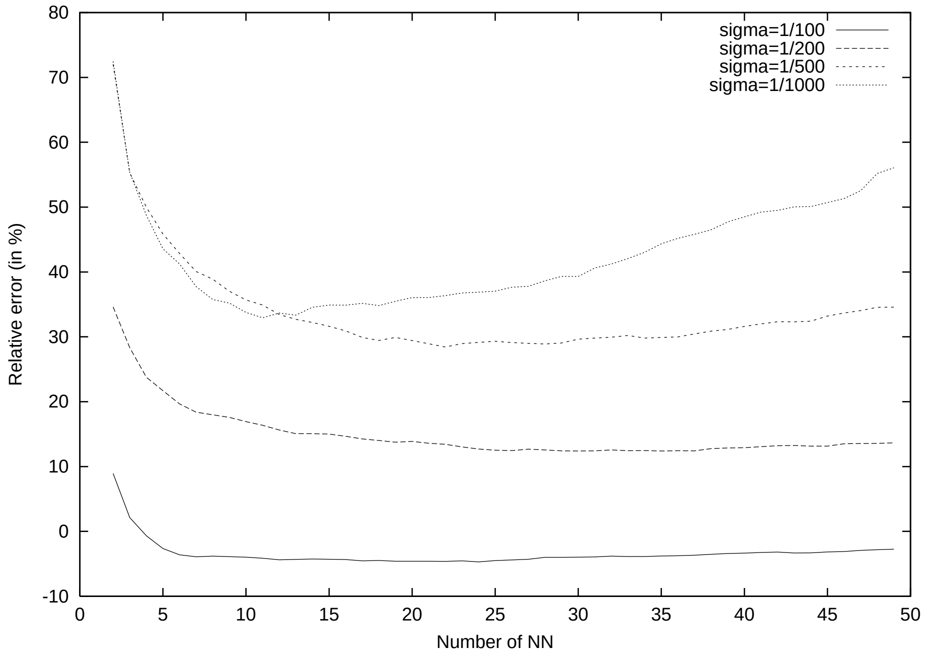


Fig. 6. Rel. error of r_{expected} (LANDSAT).

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_{expected} 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. As a comparison, the statically computed radius $d_{nn}(k)$ bottomed out at about 400% relative error. This clearly shows the advantage of query-dependent radius estimation.

6. Upper Bounding the NN-radius

In the previous sections, we saw how a query-dependent NN-radius estimate can be obtained via fractal dimensionalities and sampling. However, since this estimate may also underestimate the correct radius, its application can lead to false misses. If this is not acceptable, as is the case in our upcoming applications (cf. Secs. 7 and 8), an upper bound on the radius has to be obtained. This section shows how a close upper bound on the NN-radius can be obtained if an approximate NN-radius is known and if the underlying index structure is page-based.

6.1. *Upper bound computation*

The basic idea behind the upper bound computation is shown in Fig. 7. Assume, we are given the radius estimate as shown (labelled with "approx. NN-sphere"). Furthermore assume that the index structure distributed the dataset (indicated as dots) into four pages (labelled with A, B, C, and D). By reading all points from

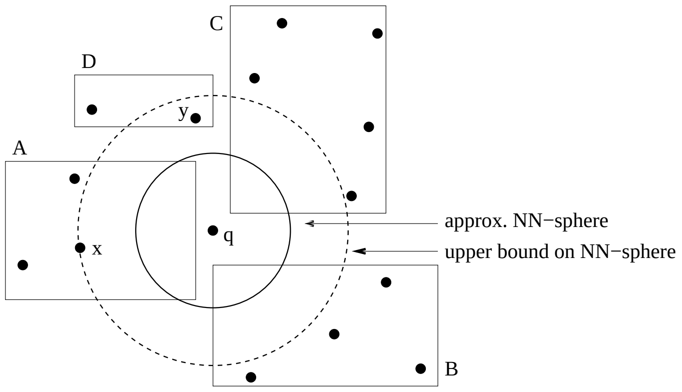


Fig. 7. Upper Bound Computation.

the pages that are intersected by the radius estimate (in this case, A, B, and C), we can determine the closest point in these pages (labelled with x). The distance to this point is an upper bound on the 1-NN radius (shown as "upper bound on NN-sphere"). Note that it is not necessarily the smallest upper bound (as point y shows).

Let us describe this estimation more formally now for arbitrary values of k. Let P be the set of index pages intersected by the approximate k-NN-sphere. 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. If $C_{\text{eff}} \cdot |P| \geq k$, we can simply pick q's k closest points from the pages in P and calculate the smallest radius that covers them. We will denote this second radius estimate by r_{upper}. Obviously, r_{upper} is larger than the real radius since the corresponding query sphere centered at q covers k points read from the pages, but not necessarily the k globally closest ones. If $C_{\text{eff}} \cdot |P| < k$, we pick the smallest distance between q and the points in the data sample (which is kept in memory) as r_{upper}.

6.2. *Quality of the upper bound*

In order to examine the tightness of r_{upper}, we performed some experiments on real data. Figure 8 shows the results for the LANDSAT dataset and an R-tree index structure. The graph was obtained as follows. First, 500 random query points were

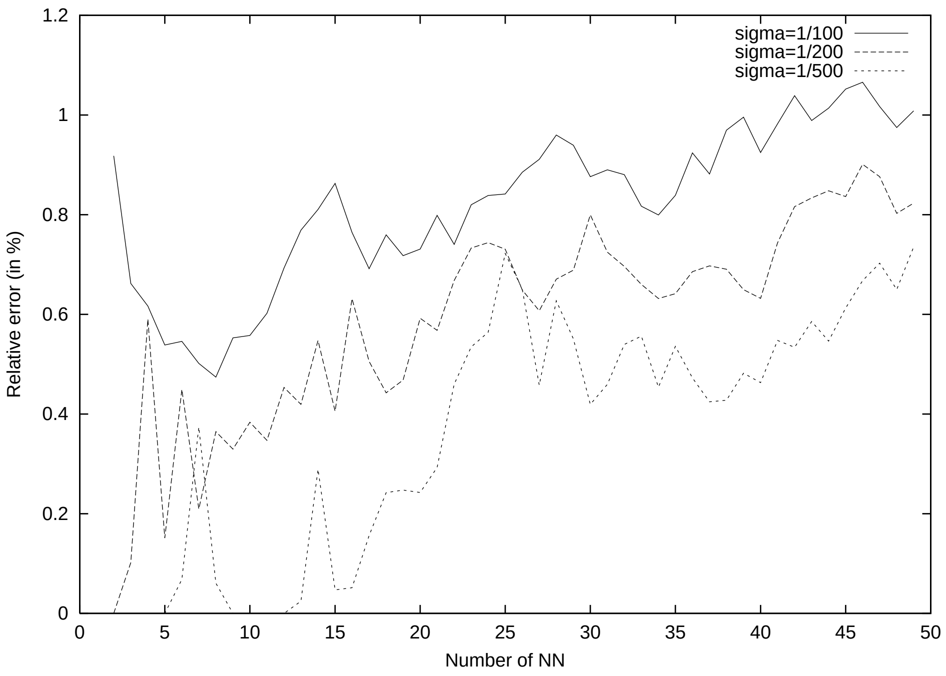


Fig. 8. Relative error of r_{upper} (LANDSAT dataset).

picked. For each query point q, we computed r_{upper} as described above by choosing r_{expected} to be the initial radius estimate. We then computed the relative error (the amount of overestimation) of r_{upper} and plotted the averages for varying k-values. As can be seen, with increasing k, the error increases slightly but it stays overall below 1.2%. This shows that the points read from the pages touched by the initial radius estimate produce a very accurate upper bound.

7. Application 1: Accelerated k-NN Query Algorithm

In this section, we present the first application of our query-dependent radius estimators. It aims at accelerating k-NN query processing by reducing the number of random disk accesses.

7.1. *The access optimal algorithm*

Before going into the details, let us first revisit the optimal k-NN query algorithm for page-based tree index structures by Hjaltason and Samet.[1] Their algorithm works as follows (cf. Fig. 9). First, a priority queue is initialized with the root page of the tree (Step 1). This queue is sorted by the MINDIST between q and the page. For a query q, an element p is removed from the queue whose MINDIST is closest to q (Step 2). If this element was a point, it is reported as a NN (Step 4). Otherwise, the element was a page. In this case, the page has to be fetched from disk and the elements stored in it are sorted into the waiting queue by their MINDIST. If the page is an inner page, the elements are typically pointers to child pages. If it is a leaf page, the elements are data points.

We assume in the following that all levels of an index tree are stored in memory. Or, in other words, only the leaf level pages have to be fetched from disk. This means that Step (3) causes only disk I/O if p was a leaf page. From a disk access point of view, the whole algorithm then reduces to the following: sort all leaf pages by their MINDIST from q and fetch them from disk in that order while keeping a list of the current k closest NNs found in the read pages. The algorithm terminates as soon as the most distant NN in the list is closer to q than the next leaf page to be read. Hjaltason and Samet proved that this algorithm is optimal in that it accesses

```
Input:    Query point $q$, number of NNs $k$
Output:   $k$ NNs of $q$

(1) Let $queue := \{root\}$.
(2) Pick page or point $p$ from $queue$
    which has smallest MINDIST from $q$.
(3) If $p$ is a page, read $p$ into memory
    and place its children in $queue$.
(4) Otherwise, report $p$ as a NN.
(5) If $k$ NNs were reported, stop. Otherwise, goto (2).
```

Fig. 9. The access optimal k-NN query algorithm.

a minimal number of pages, namely only the pages intersected by the k-NN query sphere. We will refer to this optimality as *access optimal.*

It is clear that especially for high dimensional data, the pages close to q can be scattered widely over the disk. This causes a large amount of seek operations during the query processing. If all pages to be read during a NN query would be known in advance, these expensive seeks could be avoided. We refer to an algorithm that minimizes the overall response time as *response time optimal* algorithm. Our new accelerated algorithm reduces the response time significantly by utilizing our new radius estimators. As we will see, its performance is very close to a (hypothetical) response time optimal k-NN query algorithm.

Before we discuss this new algorithm, we give the time complexity of the access optimal algorithm for a single query:

$$\begin{aligned}\text{cost}_{\text{NN}} &= L \cdot (t_{\text{seek}} + t_{\text{xfer}} + C \cdot t_{\text{cpu}}) + t_{\text{output}} \\ &= L \cdot t_{\text{seek}} + L \cdot t_{\text{xfer}} + L \cdot C \cdot t_{\text{cpu}} + t_{\text{output}} \qquad (5)\end{aligned}$$

where L denotes the number of leaf pages to be read, C denotes the number of points in one page, and t_{seek}, t_{xfer}, t_{cpu}, and t_{output} denote the disk seek time, page transfer time, CPU time for one distance calculation, and time to output the NNs, respectively. This formula assumes that each page has to be fetched separately, for the reason given above.

7.2. *The new algorithm*

This basic idea of our new k-NN query algorithm is as follows. For every incoming NN-query, we compute r_{expected} (cf. Sec. 4) as 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, we can return the k closest ones as the result set. If the first estimate was too small, with less than k points were in the range, we need to increase the search radius and perform another range query. Since we have already a first radius estimate, we can use r_{upper} (cf. Sec. 6) as this second estimate. Since r_{upper} is an upper bound on the real radius, we read at least k points during the second range query. 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). Figure 10 gives the algorithm in more detail. Note that a simple modification yields an approximate k-NN query algorithm with less I/O cost. Instead of performing a second range query, we can stop the algorithm after Step (3) and simply report the k closest points encountered during the first range query. As we saw in Sec. 5.2, for typical applications, r_{expected} is already close to the real radius. Therefore, the quality of the approximate NNs can be expected to be high.

```
Input:    Query point $q$, number of NNs $k$
Output:   $k$ NNs of $q$

(1) Perform a $k$-NN query on sample in memory;
    this yields a $k$-NN-radius $r_{\text{sample}}$.
(2) Let $r_{\text{expected}} := r_{\text{sample}} \cdot \frac{d_{nn}^{\text{cutoff}}(k)}{d_{\text{sample}-nn}^{\text{cutoff}}(k)}$.
(3) Perform a range query with radius
    $r_{\text{expected}}^{\text{cutoff}}$ 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 $r_{\text{upper}}$ enclosing them.
(6) Perform a modified range query around $q$
    with radius $r_{\text{upper}}$ 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.
```

Fig. 10. The accelerated k-NN query algorithm.

Before being able to run these algorithms for a given dataset, we need to 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 with the box counting algorithm as discussed in Sec. 3.1. Note that this cost has to be paid only once and is therefore amortized over time.

The time complexity of the accelerated algorithm for a single query is

$$\begin{aligned}\text{cost}_{\text{AccNN}} &= \frac{|P|}{B} \cdot (t_{\text{seek}} + B \cdot t_{\text{xfer}} + B \cdot C \cdot t_{\text{cpu}}) \\ &\quad + \frac{|P'|}{B} \cdot (t_{\text{seek}} + B \cdot t_{\text{xfer}} + B \cdot C \cdot t_{\text{cpu}}) + t_{\text{output}} \\ &= \frac{|P| + |P'|}{B} \cdot t_{\text{seek}} + (|P| + |P'|) \cdot t_{\text{xfer}} + (|P| + |P'|) \cdot C \cdot t_{\text{cpu}} + t_{\text{output}}\,,\end{aligned} \tag{6}$$

where B denotes the average number of pages that can be read as a bulk. In case the radius estimation is error-free, we have $|P| + |P'| = L$. When comparing to Eq. 5, it is clear that the cost savings stem from the reduced amount of seek operations due to the bulk read operations.

Disk Page Read Strategy. Only Steps (3) and (6) of our accelerated k-NN query algorithm induce disk I/O due to the two range queries. The range query in Step (3) is a regular range query provided by the indexing system. 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. Note that it is possible to perform Step (6) with a regular range query without affecting the correctness of the algorithm but the modified version reduces the amount of unnecessary page reads and thereby leads to lower I/O cost. In our experiments, we make use of the heuristic suggested by Seeger *et al.*[20] to minimize the I/O cost for reading a set of disk pages during a range query execution.

7.3. *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[h]) and a real dataset (LANDSAT). For each dataset we ran 100 k-NN queries where k varied between 10 and 50 and we measured the amount of seeks and page transfers. All queries are density-biased, i.e. query points are picked with higher probability from a region with higher density. The sampling rate is always 1/100. For the underlying harddisk we assume a 20 MB/s transfer rate and an average seek time of 10 ms. This, together with the measured seek and transfer numbers, is then used to compute the average query response time. All experiments were conducted on a prototype implementation.

We report here only on the results for the X-tree[21] index structure (a discussion on other index structures (such as R-tree[22] and VA-file[23]) can be found in the technical report[24]). For comparison, we use the optimal NN-query algorithm proposed by Hjaltason and Samet.[1] The index page capacity is 8 KB and the upper part of the index tree is kept in memory. Therefore, only leaf page accesses cause disk I/O.

In the next paragraphs, we compare the performance of the access optimal NN-query algorithm with our accelerated version, a hypothetical algorithm with perfect radius estimator, and the linear scan. The hypothetical algorithm "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. It provides a lower bound of our algorithm's response time. The linear scan reads all pages in a linear fashion and therefore requires no seek operations. It is therefore a good point of reference for judging the impact of filtering out pages by our radius estimates.

Results for Synthetic Data. Figure 11 shows the results for the X-tree and 8-dimensional uniform data. For each k-value, we show four I/O costs: for the access optimal 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. 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

[h]The uniform dataset consists of 100 000 uniformly distributed 8-dimensional points.

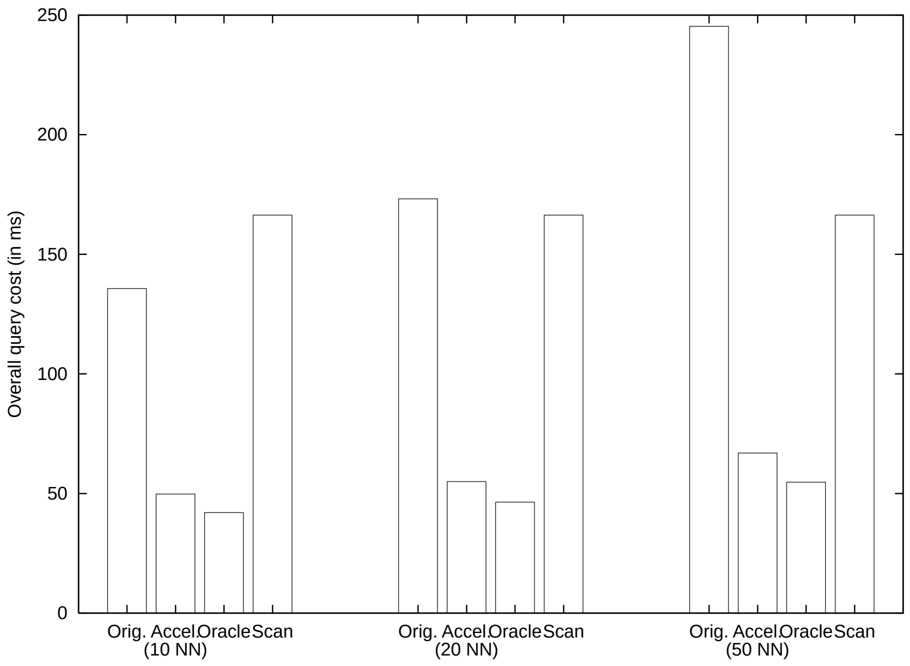


Fig. 11. Overall query cost (UNIFORM).

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.

Results for Real Data. The experimental results for the LANDSAT dataset can be found in Fig. 12. Our accelerated query algorithm outperforms the well-tuned X-tree 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 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.

8. Application 2: Accelerated Disk-based All-NN Query Algorithm

8.1. *The block-nested-loops join algorithm*

Assume we are given two point datasets, DS_1 and DS_2. For each point in DS_1 we want to determine its NN among all points in DS_2. This problem can be easily generalized to all-k-NN queries.

One straightforward way of answering such queries is by performing $|DS_1|$ regular NN queries on DS_2 (which may then be accelerated using our technique

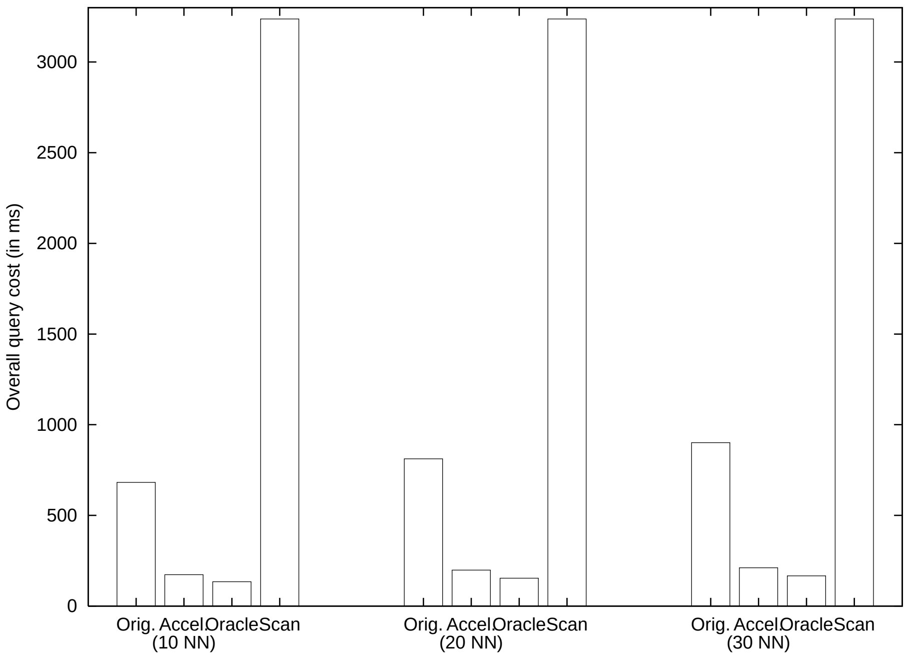


Fig. 12. Overall query cost (LANDSAT).

presented in the last section). However, this solution causes a high number of disk seeks for high-dimensional datasets since typical caches are not large enough to store common pages between queries. It would be beneficial to perform these operations in bulks.

One way of achieving this is by utilizing a block-nested-loops join algorithm[25] since the all-NN problem can be viewed as the problem of joining DS_1 and DS_2 with the join condition being $\sigma(x, y) =$ "x has y as a NN" where $x \in DS_1$ and $y \in DS_2$. The pseudocode for the block-nested-loops join algorithm is given in Fig. 13. The algorithm consists mainly of two nested loops, one reading pages from DS_1 and one reading pages from DS_2. These page reads are performed in a sequential fashion in

```
Input:    Dataset $DS_1$, dataset $DS_2$
Output:   NNs of all $q \in DS_1$ taken from $DS_2$

(1) Scan $DS_1$ in chunks of size $M/2$ (half of the buffer);
    for each chunk of $DS_1$-pages:
(2)     Read $M/2$ pages from $DS_2$ (fill other half);
        for each chunk of $DS_2$-pages:
(3)         Update NNs for each point in the $DS_1$-pages.
(4)     Output NNs for all points in the $DS_1$-pages.
(5) Stop.
```

Fig. 13. Block-nested-loops join algorithm.

order to reduce disk seeks. We assume that a buffer of size M (in number of pages) is provided. The algorithm fills the first half with DS_1-pages and the second half with DS_2-pages. For each point in a DS_1-page we store additionally its best NNs so far. This information is updated in Step (3).

The time complexity of this algorithm is

$$\begin{aligned}\mathrm{cost}_{\mathrm{Join}} &= \frac{L_1}{M/2} \cdot \Bigg(t_{\mathrm{seek1}} + \frac{M}{2} \cdot t_{\mathrm{xfer1}} \\ &\quad + \frac{L_2}{M/2} \cdot \left(t_{\mathrm{seek2}} + \frac{M}{2} \cdot t_{\mathrm{xfer2}} + \frac{M^2}{4} \cdot C^2 \cdot t_{\mathrm{cpu}} \right) + \frac{M}{2} \cdot t_{\mathrm{output}} \Bigg) \\ &= 2 \cdot \frac{L_1}{M} \cdot t_{\mathrm{seek1}} + 4 \cdot \frac{L_1 \cdot L_2}{M^2} \cdot t_{\mathrm{seek2}} + L_1 \cdot t_{\mathrm{xfer1}} \\ &\quad + 2 \cdot \frac{L_1 \cdot L_2}{M} \cdot t_{\mathrm{xfer2}} + L_1 \cdot L_2 \cdot C^2 \cdot t_{\mathrm{cpu}} + L_1 \cdot t_{\mathrm{output}}\,, \end{aligned} \tag{7}$$

where L_1 and L_2 denote the number of pages in DS_1 and DS_2, respectively. $t_{\mathrm{seek}i}$ and $t_{\mathrm{xfer}i}$ denote the seek and transfer time of disk i.

8.2. *The new algorithm*

In this section, we show how our new radius estimators can be used to accelerate the block-nested-loops join algorithm for answering all-NN queries. The basic idea is as follows: instead of estimating the NN-radius for a single point, we estimate it for all points in the DS_1-pages stored in the buffer. Once we have calculated the NN-radius-estimates for all points of a page of DS_1, we can predict which pages of DS_2 have to be accessed in order to compute the join. This is where the I/O cost is reduced significantly as compared to a block-nested-loops join algorithm which accesses *every* page of DS_2 for a page of DS_1.

Let us discuss the new algorithm in more depth. It is given in Fig. 14. In Steps (1) and (2), a so-called *influence region* $R_{\mathrm{expected}}(P)$ is computed for every page P in DS_1. This can be seen as a generalization of the computation of r_{expected} for the accelerated k-NN query algorithm. The influence region can be regarded as the union of the expected NN-spheres of all points in P, as shown in Fig. 15. In this example, page P contains three points. The influence region is shown as the shaded area. Two of the DS_2-pages (shown as dashed boxes) are intersected by P's influence region and have to be retrieved in Step (4) of the algorithm. Page A would not be retrieved at this time.

In case DS_2-pages that have to be retrieved are less than $M/2$ pages apart on disk, they can be read together as in Step (4). This helps reducing disk seeks. In our implementation, we access the $M/2$ pages around the intersected DS_2-page in disk-placement order. Since most index structures try to place pages nearby on the disk that are spatially close in the data space, pages stored before and after the

```
Input:    Dataset $DS_1$, dataset $DS_2$
Output:   NNs of all $q \in DS_1$ taken from $DS_2$

(1)  Scan $DS_1$ in chunks of size $M$ (whole buffer);
     for each chunk of $DS_1$-pages:
(2)      Compute expected influence region $R_{\text{expected}}(P)$
         for each $DS_1$-page $P$.
(3)  Scan $DS_1$ in chunks of size $M/2$ (half of the buffer);
     for each chunk of $DS_1$-pages:
(4)      Read $M/2$ consecutive pages from $DS_2$
         that intersect at least one influence region
         of the $DS_1$-pages (fill other half of $M$);
         for each chunk of $DS_2$-pages:
(5)          Update NNs for each point in the $DS_1$-pages.
(6)          Update $R_{\text{expected}}(P)$ for each $DS_1$-page $P$.
(7)      Read all pages (at most $M/2$ at a time)
         of $DS_2$ that intersect at least one influence
         region of the $DS_1$-pages (fill other half of $M$);
         for each chunk of $DS_2$-pages:
(8)          Update NNs for each point in the $DS_1$-pages.
(9)      Output NNs for all points in the $DS_1$-pages.
(10) Stop.
```

Fig. 14. Accelerated all-NN query algorithm.

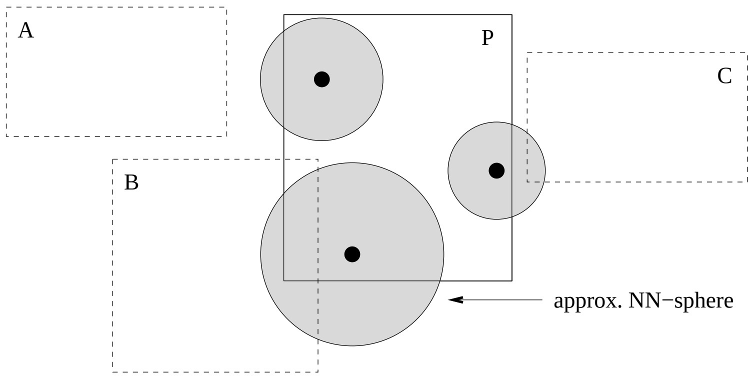


Fig. 15. Influence region of page P.

intersected page have a high probability to contain points that are also close by. This leads to better updated estimates in Step (6) of the algorithm.

After $M/2$ DS_2-pages were read in Step (4), the NN-information is updated in Step (5) similar to the block-nested-loops join algorithm. Then the points found in the $M/2$ DS_2-pages are used to improve $R_{\text{expected}}(P)$ for each DS_1-page in the buffer. This can be seen as the equivalent to Step (5) of our accelerated k-NN query algorithm. Similar to there, the updated $R_{\text{expected}}(P)$ represents an upper bound on the correct influence region of P.

The updated $R_{\text{expected}}(P)$ is used in Step (7) to read the pages from DS_2 that are intersected by this upper bound influence region. In the remainder of the algorithm, the NN-info of all points in DS_1-pages are updated and output once all DS_2-pages are processed.

An important observation is that *the accelerated all-NN search algorithm can never perform more than twice as many page accesses as the block-nested-loops join algorithm.* This can be seen as follows: in the worst case, all pages of DS_2 are accessed in Line (4) and again in Line (7), since Step (4) reads additional (possibly unnecessary) pages that may be refetched in Step (7).

The overall time complexity of the new algorithm is

$$\begin{aligned}\text{cost}_{\text{AccJoin}} &= \frac{L_1}{M} \cdot (t_{\text{seek1}} + M \cdot t_{\text{xfer1}} + M \cdot C \cdot S \cdot t_{\text{cpu}}) + \frac{L_1}{M/2} \cdot \left(t_{\text{seek1}} + \frac{M}{2} \right.\\ &\quad \times t_{\text{xfer1}} + \frac{L_2'}{M/2} \cdot \left(\frac{M/2}{B} \cdot t_{\text{seek2}} + \frac{M}{2} \cdot t_{\text{xfer2}} + \frac{M^2}{4} \cdot C^2 \cdot t_{\text{cpu}} \right)\\ &\quad \left. + \frac{L_2''}{M/2} \cdot \left(\frac{M/2}{B} \cdot t_{\text{seek2}} + \frac{M}{2} \cdot t_{\text{xfer2}} + \frac{M^2}{4} \cdot C^2 \cdot t_{\text{cpu}} \right) + \frac{M}{2} \cdot t_{\text{output}} \right)\\ &= 3 \cdot \frac{L_1}{M} \cdot t_{\text{seek1}} + 2 \cdot \frac{L_1}{M} \cdot \frac{L_2' + L_2''}{B} \cdot t_{\text{seek2}} + 2 \cdot L_1 \cdot t_{\text{xfer1}}\\ &\quad + 2 \cdot \frac{L_1}{M} \cdot (L_2' + L_2'') \cdot t_{\text{xfer2}} + L_1 \cdot (L_2' + L_2'') \cdot C^2 \cdot t_{\text{cpu}} + L_1 \cdot t_{\text{output}} \,,\end{aligned} \tag{8}$$

where S denotes the size of the sample used in Step (2) and L_2' and L_2'' denote the average number of pages to be read from DS_2 in Steps (4) and (7), respectively.

In order to compare Eq. 7 and Eq. 8, let us assume DS_1 and DS_2 are located on the same type of disk, i.e. $t_{\text{seek1}} = t_{\text{seek2}}$ and $t_{\text{xfer1}} = t_{\text{xfer2}}$. Then the seek times are

$$\text{cost}_{\text{Join}}^{\text{seek}} = \frac{L_1}{M}\left(2 + 4\frac{L_2}{M}\right) \leftrightarrow \text{cost}_{\text{AccJoin}}^{\text{seek}} = \frac{L_1}{M}\left(3 + 2\frac{L_2' + L_2''}{B}\right).$$

Since the number of pages read as a bulk B is typically close to M and $L_2' + L_2'' \ll L_2$, the seek time can be expected to drop.

The page transfer times are

$$\text{cost}_{\text{Join}}^{\text{xfer}} = L_1\left(1 + 2\frac{L_2}{M}\right) \leftrightarrow \text{cost}_{\text{AccJoin}}^{\text{xfer}} = L_1\left(2 + 2\frac{L_2' + L_2''}{M}\right).$$

Again, since $L_2' + L_2'' \ll L_2$, the transfer time can be expected to drop.

Finally, the time spent in distance calculations is

$$\text{cost}_{\text{Join}}^{\text{cpu}} = L_1 L_2 C^2 \leftrightarrow \text{cost}_{\text{AccJoin}}^{\text{cpu}} = L_1(L_2' + L_2'')C^2 \,.$$

With the same reasoning, this cost can be expected to drop as well.

8.3. *Experimental results*

For experimental evaluation, we ran two experiments: on two-dimensional synthetic uniform data, and on 60-dimensional real data. The uniform datasets consist of 10 000 points each and the real datasets consist of approximately 13 000 points each extracted randomly from the LANDSAT dataset. In the case of joins, the overall running times are not necessarily I/O-bound. Due to the high number of distance computations required, the CPU cost becomes an important factor. Hence, we include both costs in our comparisons. For the uniform data, our accelerated join algorithm dropped the overall response time from about 4 minutes to less than 1 minute. In the LANDSAT case, the response time dropped from 150 minutes to about 40 minutes. The buffer size M (which was varied between 3 and 25% of all pages) had no noticeable impact on this drop. The results are shown in Figs. 16 and 17.

The largest percentage of the cost savings stems from the reduced number of distance computations performed by our new algorithm. For the lower-dimensional data, only 1/5 of all distance computations have to be performed. For the higher-dimensional data, only 1/4 of the computations are necessary. This shows that the pruning caused by our influence region estimation is very effective. The I/O cost is also reduced in both cases but since distance computations are very costly for high-dimensional data, the algorithm becomes CPU-bound in such settings.

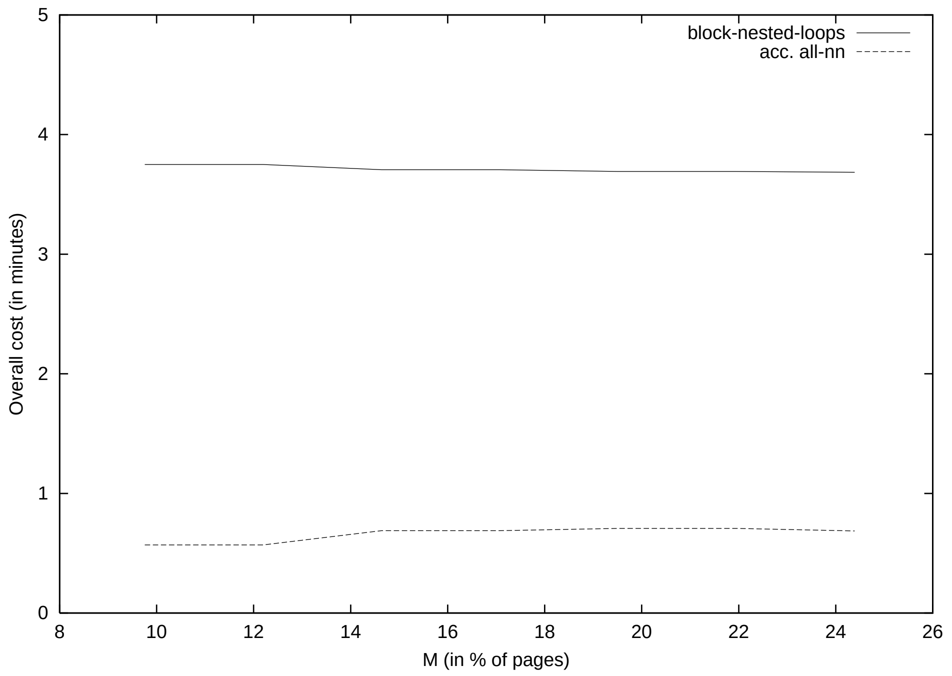


Fig. 16. Overall query cost (UNIFORM).

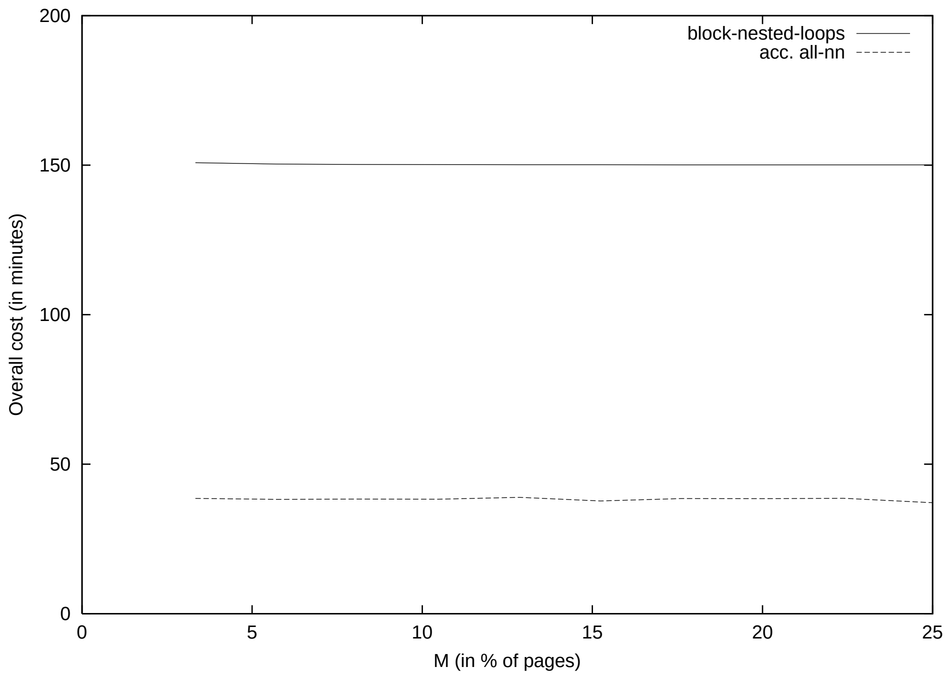


Fig. 17. Overall query cost (LANDSAT).

9. Conclusions

We showed in this paper how query-dependent query range estimates can be used to accelerate two important query algorithms. As a major contribution, we introduced the notion of query-dependent query range estimates and gave an algorithm for their computation based on sampling. Our analysis indicates that the estimation error for the expected radius is below 14% and that errors in the fractal dimensionality estimation have only minor impact on the accuracy. For the upper bound estimate, the observed error was even lower, namely below 1%.

In the experimental section, we introduced two new algorithms that we accelerated using the estimates discussed in the first part of the paper: one for answering k-NN queries, and one for answering all-NN queries. Our new k-NN query algorithm accelerates a bulkloaded X-tree index structure by a factor of nearly five. Even the VA-file, which does not cluster points in pages, benefits from our acceleration. The all-NN-query algorithm we presented outperforms a block-nested-loops join algorithm by a factor of 3–4. Surprisingly, our technique was able to accelerate significantly query algorithms proposed as part of index structures.

We want to emphasize that our technique is not to be understood as an improved X-tree or another index structure. It should rather be seen as a general technique for accelerating existing indexing schemes.

Acknowledgments

This work is partially supported by NSF under grants EIA-0080134, EIA-9986057, IIS-9877142, ANI-0123985, and NSFIIS98-17432.

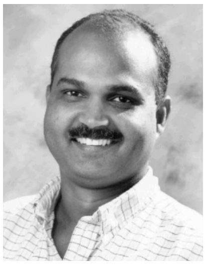
References

1. G. R. Hjaltason and H. Samet, "Ranking in spatial databases," in *Advances in Spatial Databases — Fourth Int. Symp.* (1995), pp. 83–95.
2. C. Faloutsos and I. Kamel, "Beyond uniformity and independence: Analysis of R-trees using the concept of fractal dimension," in *Proc. ACM Symp. Principles of Database Syst.* (1994), pp. 4–13.
3. A. Papadopoulos and Y. Manolopoulos, "Performance of nearest neighbor queries in R-trees," in *Proc. Int. Conf. Database Theory*, *Lecture Notes in Computer Science* **1186**, 394–408 (1997).
4. F. Korn, B.-U. Pagel and C. Faloutsos, "Deflating the dimensionality curse using multiple fractal dimensions," in *Proc. Int. Conf. Data Engineering* (2000).
5. C.-M. Chen and Yibei Ling, "A sampling-based estimator for top-k query," in *Proc. Int. Conf. Data Engineering* (2002).
6. 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. Data Engineering* (2000).
7. P. Ciaccia and M. Patella, "Bulk loading the M-tree," in *9th Australasian Database Conference* (1998), pp. 15–26.
8. Y. Theodoridis and T. K. Sellis, "A model for the prediction of R-tree performance," in *Proc. ACM Symp. Principles of Database Systems* (1996), pp. 161–171.
9. S. Acharya, V. Poosala and S. Ramaswamy, "Selectivity estimation in spatial databases," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (1999), pp. 13–24.
10. F. Korn, T. Johnson and H. V. Jagadish, "Range selectivity estimation for continuous attributes," in *Proc. Int. Conf. Scientific and Statistical Database Management* (1999), pp. 244–253.
11. J. Jin, N. An and A. Sivasubramaniam, "Analyzing range queries on spatial data," in *Proc. Int. Conf. Data Engineering* (2000).
12. C. A. Lang and A. K. Singh, "Modeling high-dimensional index structures using sampling," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (2001).
13. T. Brinkhoff, H. Kriegel and B. Seeger, "Efficient processing of spatial joins using R-trees," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (1993).
14. Y.-W. Huang, N. Jing and E. A. Rundensteiner, "Spatial joins using r-trees: Breadth-first traversal with global optimizations," in *Proc. Int. Conf. Very Large Data Bases* (1997), pp. 396–405.
15. N. Koudas and K. C. Sevcik, "High dimensional similarity joins: Algorithms and performance evaluation," in *Proc. Int. Conf. Data Engineering* (1998), pp. 466–475.
16. C. Böhm, B. Braunmüller, F. Krebs and H.-P. Kriegel, "Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (2001).
17. G. R. Hjaltason and H. Samet, "Incremental distance join algorithms for spatial databases," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (1998), pp. 237–248.

18. B. Braunmüller, M. Ester, H.-P. Kriegel and J. Sander, "Efficiently supporting multiple similarity queries for mining in metric databases," in *Proc. Int. Conf. Data Engineering* (2000), pp. 256–267.
19. A. Belussi and C. Faloutsos, "Estimating the selectivity of spatial queries using the 'correlation' fractal dimension," in *Proc. Int. Conf. Very Large Data Bases* (1995), pp. 299–310.
20. B. Seeger, P.-Å. Larson and R. McFayden, "Reading a set of disk pages," in *Proc. Int. Conf. Very Large Data Bases* (1993), pp. 592–603.
21. S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The X-tree: An index structure for high-dimensional data," in *Proc. Int. Conf. Very Large Data Bases* (1996), pp. 28–39.
22. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD Int. Conf. Management of Data* (1984), pp. 47–57.
23. R. Weber and S. Blott, "An approximation based data structure for similarity search," Technical Report 24, ESPRIT project HERMES (no. 9141), October 1997.
24. C. A. Lang and A. K. Singh, "Accelerating high-dimensional nearest neighbor queries," Technical Report CS-TR-0204, University of California at Santa Barbara, January 2002.
25. R. Ramakrishnan and J. Gehrke, *Database Management Systems* (2nd Edition) (McGraw-Hill, 2000).

Christian Lang received the Diploma in Computer Science from the Technical University of Munich, Germany, in 1996.

In 1997, he joined the University of California at Santa Barbara as a PhD student. His research interests are high-dimensional indexing and performance prediction. He is a student member of the ACM, ACM SIGMOD, and IEEE.



Ambuj Singh received a Masters in Computer Science from Iowa State University in 1984 and a PhD in Computer Science from the University of Texas in 1989.

He joined the University of California at Santa Barbara in 1989, where he is currently a Full Professor. His research interests are in the areas of multimedia databases, distributed computing, and bioinformatics.