An Efficient Cost Model for Optimization of Nearest Neighbor Search in Low and Medium Dimensional Spaces

Yufei Tao, Jun Zhang, Dimitris Papadias, and Nikos Mamoulis

Abstract—Existing models for nearest neighbor search in multidimensional spaces are not appropriate for query optimization because they either lead to erroneous estimation or involve complex equations that are expensive to evaluate in real-time. This paper proposes an alternative method that captures the performance of nearest neighbor queries using approximation. For uniform data, our model involves closed formulae that are very efficient to compute and accurate for up to 10 dimensions. Further, the proposed equations can be applied on nonuniform data with the aid of histograms. We demonstrate the effectiveness of the model by using it to solve several optimization problems related to nearest neighbor search.

Index Terms—Information storage and retrieval, selection process.

1 Introduction

Given a multidimensional point data set S, a k *nearest neighbor* (NN) query returns the k points of S closest to a query point according to a certain distance function. NN queries constitute the core of similarity search in spatial [24], [18], multimedia databases [26], time series analysis [15], etc. Accurate estimation of NN performance is crucial for query optimization, e.g., it is well-known that the efficiency of indexed-based NN algorithms degrades significantly in high-dimensional spaces so that a simple sequential scan often yields better performance [6], [31], [7]. As shown later, a similar problem also exists in low and medium dimensionality for queries returning a large number of neighbors. Thus, the ability to predict the cost enables the query optimizer to decide the threshold of dimensionality or k (i.e., the number of neighbors retrieved) above which sequential scan should be used. Further, NN queries are often components of complex operations involving multiple predicates (e.g., "find the 10 nearest cities to New York with population more than 1M"), in which case NN performance analysis is indispensable for generating alternative evaluation plans. The necessity of NN analysis is further justified in [9], [28], which show that an efficient model can be used to tune the node size of indexes in order to reduce the number of random disk accesses and decrease the overall running time.

As surveyed in the next section, the existing models are not suitable for query optimization because they either suffer from serious inaccuracy or involve excessively complex integrals that are difficult to evaluate in practice. Even more seriously, their applicability to nonuniform data is limited because 1) they typically assume *biased queries* (i.e., the query distribution is the same as that of the data), and 2) they are able to provide only a single estimate, which corresponds to the average performance of all possible queries. However, queries at various locations of the data space have different behavior, depending on the data characteristics in their respective vicinity. As a result, the average performance cannot accurately capture all individual queries.

A practical model for NN search should be closed (i.e., it should not involve complex integrals, series, etc.), easy to compute, precise, and able to provide a "tailored" estimate for each query. Motivated by this, we deploy a novel approximation method which aims at high precision with limited evaluation overhead. An important merit of our model is that it permits the application of conventional multidimensional histograms for individual NN queries on nonuniform data. As a second step, we apply the proposed formulae to several important query optimization problems.

This paper focuses on *vector* data spaces of low or medium dimensionality[1] (up to 10 dimensions) and Euclidean distance (i.e., the L_2 norm) due to its popularity. The next section introduces NN search algorithms, reviews the existing cost models, and elaborates on their shortcomings. Section 3 presents our model, first on uniform data and then extending the solution to arbitrary distributions. Section 4 demonstrates the applicability of the new model for query

- *Y. Tao is with the Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. E-mail: taoyf@cs.cityu.edu.hk.*
- *J. Zhang is with the Division of Information Systems, Computer Engineering School, Nanyang Technological University, Singapore. E-mail: jzhang@ntu.edu.sg.*
- *D. Papadias is with the Department of Computer Science, Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong. E-mail: {zhangjun, dimitris}@cs.ust.hk.*
- *N. Mamoulis is with the Department of Computer Science and Information Systems, Hong Kong University, Pokfulam Road, Hong Kong. E-mail: nikos@csis.hku.hk.*

Manuscript received 21 May 2002; revised 7 June 2003; accepted 26 June 2003.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number 116604.

1. From a practical point of view, analysis of NN search in high dimensional spaces is less important because index-based algorithms are usually outperformed by sequential scan [6], [3].

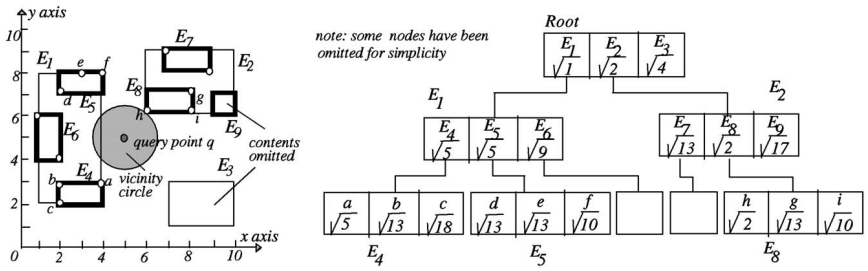


Fig. 1. Example R*-tree and nearest neighbor query.

optimization, and Section 5 contains an extensive experimental evaluation to prove its efficiency. Finally, Section 6 concludes the paper with directions for future work.

2 Related Work

Section 2.1 introduces indexed-based NN algorithms and discusses the related problem of distance browsing. Then, Section 2.2 surveys the existing analysis and cost models.

2.1 Algorithms for kNN Queries

Following the convention in the literature, throughout the paper, we assume the R*-tree [4] as the underlying index, but our discussion generalizes to other data partitioning access methods (such as X-trees [5], A-trees [27], etc.). Fig. 1 shows a data set (with data points a, b, c, ...) and the corresponding R*-tree, which clusters the objects by their spatial proximity. Each nonleaf entry of the tree is associated with a minimum bounding rectangle (MBR) that encloses all the points in its subtree.

Consider, for example, the nearest neighbor query at coordinate (5, 5), whose distances to the data points and MBRs[2] are illustrated using the numbers associated with the leaf and nonleaf entries, respectively (these numbers are for illustrative purpose only and are not actually stored in the tree). An optimal kNN algorithm only visits those nodes whose MBRs intersect the *search region or vicinity circle*[3] $\Theta(q, D_k)$ that centers at q with radius D_k equal to the distance between the query point and the kth nearest neighbor [23]. In the example of Fig. 1, $k = 1$, D_1 equals the distance between q and h, and the vicinity circle is shown in gray.

An important variation of kNN search is called *distance browsing* (or *distance scanning*) [18], where the number k of neighbors to be retrieved is not known in advance. Consider, for example, a query that asks for the nearest city of New York with population more than one million. A distance browsing algorithm first finds the nearest city c_1 of New York and examines whether the population of c_1 is more than one million. If the answer is negative, the algorithm retrieves the next nearest city c_2 and repeats this process until a city satisfying the population condition is found. The implication of such *incremental processing* is that, having obtained the k nearest neighbors, the $(k+1)$th neighbor should be computed with little overhead.

Existing nearest neighbor algorithms prune the search space following the branch-and-bound paradigm. The *depth-first* (DF) algorithm (see [24] for details) starts from the root and follows recursively the entry closest to the query point. It is, however, suboptimal and cannot be applied for incremental nearest neighbor retrieval. The best-first (BF) algorithm of [18] achieves optimal performance by keeping a heap containing the entries of the nodes visited so far. The contents of the heap for the example of Fig. 1 are shown in Fig. 2. Initially, the heap contains the entries of the root sorted according to the distances of their MBRs to the query point. At each iteration, the algorithm removes (from the heap) the entry with the smallest distance and examines its subtree. In Fig. 1, E_1 is visited first, and the entries of its child node (E_4, E_5, E_6) are inserted into the heap together with their distances. The next entry accessed is E_2 (its distance is currently the minimum in the heap), followed by E_8, where the actual result (h) is found and the algorithm terminates. The extension to k nearest neighbors is straightforward; the only difference is that the algorithm terminates after having removed k data points from the heap. BF is incremental, namely, the number of neighbors to be returned does not need to be specified; hence, it can be deployed for distance browsing.

2.2 Existing Performance Studies and Their Defects

Analysis of kNN queries aims at predicting: 1) the *nearest distance* D_k (the distance between the query and the kth nearest neighbor), 2) the query cost in terms of the number of index nodes accessed or, equivalently, the number of nodes whose MBRs intersect the search region $\Theta(q, D_k)$. The earliest models for nearest neighbor search [13], [10] consider only single $(k = 1)$ nearest neighbor retrieval assuming the L_∞ metric and $N \to \infty$, where N is the total number of points in the data set. Sproull [25]

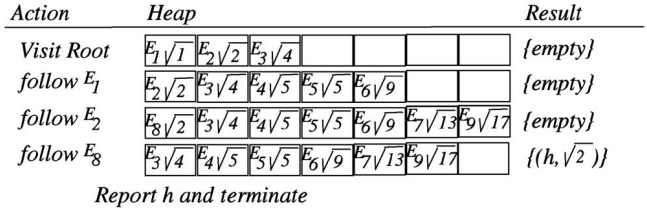


Fig. 2. Heap contents during BF.

2. The distance between a point p and an MBR r equals the minimum of the distances between p and any point in r (see [24] for a formal definition and details on the computation).

3. For dimensionality $d \geq 3$, a vicinity circle becomes a sphere or a hypercircle. In this paper, we use these terms interchangeably.

presents a formula suggesting that, in practice, N must be exponential with the dimensionality for the models of [13], [10] to be accurate. When this condition is not satisfied, these models yield considerable errors due to the so-called *boundary effect*, which occurs if the distance from the query point to its kth nearest neighbor is comparable to the axis length of the data space. The first work [1] that takes boundary effects into account also assumes the L_∞ metric. Papadopoulos and Manolopoulos [23] provide lower and upper bounds of the nearest neighbor query performance on R-trees for the L_2 norm. Boehm [3] points out that these bounds become excessively loose when the dimensionality or k increases and, as a result, they are of limited use in practice.

The most accurate model is presented by Berchtold et al. [6] and Boehm [3]. To derive the average distance D_1 from a query point q to its nearest neighbor, they utilize the fact that, for uniform data and query distributions in the unit data space U, the probability $P(q, \leq r)$ that a point falls in the vicinity circle $\Theta(q, r)$ corresponds to its volume $Vol(\Theta(q, r))$. Part of $\Theta(q, r)$, however, may fall outside the data space and should not be taken into account in computing $P(q, \leq r)$. To capture this boundary effect, $P(q, \leq r)$ should be calculated as the expected volume of the intersection of $\Theta(q, r)$ and U (i.e., averaged over all possible locations of q):

$$P(q, \leq r) = E[Vol(\Theta(q, r) \cap U)] = \int_{p \in U} Vol(\Theta(p, r) \cap U) dp. \tag{1}$$

Based on $P(q, \leq r)$, the probability $P(D_1 \leq r)$ that the nearest distance is smaller than r (i.e., there is at least one point in $\Theta(q, r)$) is represented as:

$$P(D_1 \leq r) = 1 - (1 - P(q, \leq r))^N. \tag{2}$$

The density function $p(D_1 = r)$ of $P(D_1 \leq r)$ is the derivative of $P(q, \leq r)$:

$$p(D_1 = r) = \frac{dP(D_1 \leq r)}{dr} = \frac{dP(q, \leq r)}{dr} \cdot N \cdot (1 - P(q, \leq r))^{N-1}. \tag{3}$$

Hence, the expected value of D_1 is:

$$\begin{aligned} E(D_1) &= \int_0^\infty r \cdot p(D_1 = r) dr \\ &= N \cdot \int_0^\infty r \frac{dP(q, \leq r)}{dr} \cdot (1 - P(q, \leq r))^{N-1} dr. \end{aligned} \tag{4}$$

The evaluation of the above formula, however, is prohibitively expensive, which renders the model inapplicable for query optimization. Specifically, as shown in [6] and [31], the integral in (1) must be computed using the Monte-Carlo method, which determines the volume of an object with complex shape by 1) generating a large number of points, 2) counting the number of points inside the object's boundary, and 3) dividing this number by the total number of points. Based on this, (4) is solved numerically using the trapezoidal rule as follows: First, the integral range $[0, \infty)$ is divided into several partitions, where the integral function is approximated by a trapezoid in each partition. To compute the area of each trapezoid, values of $dP(q, \leq r)/dr$ and $P(q, \leq r)$ at the end points of the corresponding partition must be evaluated (by the Monte-Carlo method). Finally, the sum of the areas of all trapezoids is taken as the value of the integral.

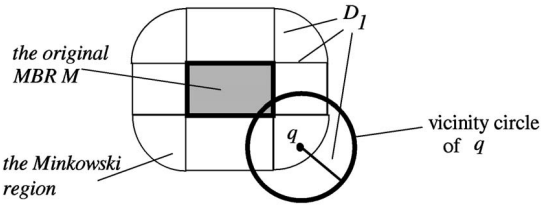


Fig. 3. The Minkowski region of M contains q.

To remedy the high cost of Monte-Carlo, Boehm [3] suggests precomputing $P(q, \leq r)$ at discrete values of r in its range $[0, d^{1/2}]$ (note that $d^{1/2}$ is the largest distance between two points in the d-dimensional space). During model evaluation, $P(q, \leq r)$ is rounded to the value of the closest precomputed r and, as a result, the expensive Monte-Carlo step can be avoided (it is pushed to the compilation time). The problem of this approach is that the number of precomputed values must be large in order to guarantee satisfactory accuracy.[4] This implies that the computed values may need to be stored on the disk in practice, in which case, evaluating the model involves disk accesses, thus compromising the evaluation cost. Extending the above analysis to D_k (i.e., the distance from the query point to the kth nearest neighbor) is relatively easy [3], but the resulting formula (5) suffers from similar evaluation inefficiency. Another problem of (4) and (5) is that they involve unsolved integrals and are, hence, difficult to deduce other properties. For example, given a query point q and distance D, it is not possible to derive how many points fall in $\Theta(q, D)$ (i.e., this requires solving k from (5) by setting $E(D_k) = D$).

$$E(D_k) = \int_0^\infty r \frac{d\left[1 - \sum_0^{k-1} \binom{N}{i} P(q, \leq r)^i (1 - P(q, \leq r))^{N-i}\right]}{dr} dr. \tag{5}$$

After deciding D_k, the second step estimates the number of node accesses, i.e., the number of nodes whose MBRs intersect $\Theta(q, D_k)$. An MBR M intersects $\Theta(q, D_k)$ if and only if its Minkowski region $\Xi(M, D_k)$, which extends M with distance D_k on all directions (see Fig. 3 for a 2D example), contains q. In the unit data space U, the intersection probability equals the volume of $Vol(\Xi(M, D_k) \cap U)$, namely, the intersection of $\Xi(M, D_k)$ and U:

$$\begin{aligned} &Vol(\Xi(M, D_k) \cap U \\ &= \int_{p \in U} \left(\begin{cases} 1 & if\ MINDIST(M, p) \leq D_k \\ 0 & otherwise \end{cases} \right) dp, \end{aligned} \tag{6}$$

where $MINDIST(M, p)$ denotes the minimum distance between an MBR M and a point p. The expected number of

4. In the experiments of [3], the precomputed information for cardinality $N = 100$K amounts to several megabytes. The size is even larger for higher N.

TABLE 1
Primary Notation Used throughout the Paper

Symbol	Description
k	the number of nearest neighbors retrieved
U	the data space
d	the dimensionality of the data space
N	the cardinality of the dataset
b	the capacity of an R-tree node
f	the average fanout of an R-tree node
h	the height of an R-tree
s_i	the average extent of a level-i node
n_i	the number of nodes at level i
P_{NAi}	the probability that a level-i node is accessed in a kNN query
D_i	the distance from the query point to the i-th nearest neighbor
$\Theta(q,r)$	the vicinity circle that centers at q and has radius r
$\mathrm{R_V}(q,r)$	the vicinity rectangle whose centroid is at q and has side length r
$\Xi(M,r)$	the Minkowski region of a rectangle with distance r
$\mathrm{R_{MINK}}(M,r)$	the approximate Minkowski rectangle with side length r
$Vol(\ )$	the volume of a geometric shape
C_V	the vicinity constant
C_{MINK}	the Minkowski constant

node accesses can be derived as the sum of $Vol(\Xi(M, D_1) \cap U)$ for all nodes. As with (4) and (5), solving (6) also requires expensive numerical evaluation and the knowledge of the node extents. Berchtold et al. [6] and Boehm [3] focus only on high dimensional spaces $(d > 10)$ where 1) nodes can split at most once on each dimension so that node extents can be either $1/2$ (for dimensions that have been split) or 1 (for the rest), and 2) the extent of a node on each dimension touches one end of the data space boundaries. These two conditions permit the use of precomputation to accelerate the evaluation, as described in [3]. However, these properties are not satisfied if the dimensionality is below 10 (as explained in the next section). Extending the precomputation method accordingly is nontrivial and not considered in [3].

The above analysis is valid only for uniform data distribution. Boehm [3] and other authors [22], [19] extend the solution to nonuniform data by computing the *fractal dimension* of the input data set. The problem of these techniques is that 1) they only deal with *biased queries* (i.e., the query distribution must follow that of the data) and, even in this case, 2) they can provide only a single estimate, which equals the average cost (of all queries), but will be used for the optimization of any individual query. As mentioned earlier, for nonuniform data sets, queries at various locations usually have different characteristics (e.g., they lead to different D_k) and, thus, applying the approaches of [3], [22], [19] may (very likely) result in inefficient execution plans.

Among others, Ciaccia et al. [12] perform an analysis similar to [6], but in the metric space. Further, Berchtold et al. [7] present a closed formula that, however, assumes that the query point lies on the diagonal of the data space. It also applies only to high-dimensional space, making the same assumptions as [3] about the node extents. It is evident from the above discussion that currently there does not exist any cost model suitable for low and medium dimensionalities (≤ 10). In the next section, we present closed formulae that overcome these problems using novel approximation techniques.

3 The Proposed Model

Traditional analysis on kNN search focuses primarily on small values of k (≤ 10), while, in many applications, it is necessary to return a larger number of neighbors. For example, in distance browsing, it is common that a large number of objects are examined (in ascending order of their distances to the query point) before one that satisfies a user's requirement is found. In this case, boundary effects cannot be ignored in estimating the query cost, even in low dimensional spaces, due to the fact that the nearest distance D_k from the query point q to its kth nearest neighbor is comparable to the extent of the data space.

The main difficulty in solving integrals (1) and (4), which capture boundary effects, lies in the computation of the intersection between a nonrectangular region (specifically, a circle in (1) or the Minkowski region in (3)) and the data space. Our analysis approximates a nonrectangular region using a rectangle with the same volume. As with the previous models, we follow a two-step method: Section 3.1 estimates the nearest distance D_k, and Section 3.2 estimates the number of nodes whose MBRs intersect the vicinity circle $\Theta(q, D_k)$, focusing on uniform data sets. In Section 3.3, we extend our approach to nonuniform data sets with the aid of histograms. Table 1 lists the symbols that will be used frequently in our discussion.

3.1 Estimating the Nearest Distance D_k

D_k satisfies the property that the vicinity circle $\Theta(q, D_k)$ is expected to contain k (out of N) points. Equivalently, for uniform data sets, this means that the expected volume $E[Vol(\Theta(q, D_k) \cap U)]$ (recall that part of the vicinity circle $\Theta(q, D_k)$ may fall outside the data space U and should not be considered) equals k/N. Solving D_k from this equation, as explained in Section 2, requires expensive numerical evaluation. Therefore, we propose to replace $\Theta(q, r)$ with a

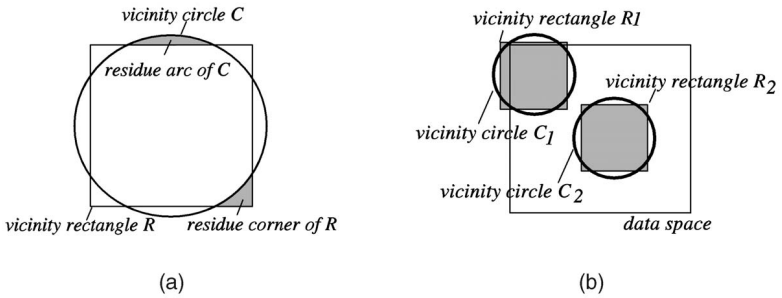


Fig. 4. Approximating vicinity circles with rectangles. (a) Residue arcs and corners. (b) Vicinity circles totally/partially in the data space.

vicinity (hyper-) rectangle $R_V(q, L_r)$ whose centroid is at q and whose extent L_r along each dimension is such that the volume of $R_V(q, L_r)$ equals that of $\Theta(q, r)$. The rationale for this choice is that, for uniform distribution, volume is the primary factor in deciding, probabilistically, the number of points in a geometric shape. Specifically, the volumes of $\Theta(q, r)$ and $R_V(q, L_r)$ are computed as:

$$\begin{aligned} &Vol(\Theta(q,r)) = \frac{\sqrt{\pi^d}}{\Gamma(d/2+1)} r^d \\ &Vol(R_V(q,L_r)) = {L_r}^d, \\ &\text{where } \Gamma(x+1) = x \cdot \Gamma(x), \Gamma(1) = 1, \Gamma(1/2)\pi = 1/2\pi. \end{aligned} \tag{7}$$

Hence, we define L_r as (by solving $Vol(\mathrm{R_V}(q,L_r)) = Vol(\Theta(q,r))$):

$$\begin{aligned} &L_r = C_V \cdot r, \text{ where } C_V \text{ is the } \textit{vicinity constant}: \\ &C_V = \frac{\sqrt{\pi}}{[\Gamma(d/2+1)]^{\frac{1}{d}}}. \end{aligned} \tag{8}$$

Using L_r, $E[Vol(\Theta(q,r) \cap U)]$ can be rewritten as (note the second equality is approximate):

$$\begin{aligned} &E[Vol(\Theta(q,r) \cap U)] \\ &= \int_{p \in U} Vol(\Theta(p,r) \cap U) dp \\ &\approx \int_{p \in U} Vol(\mathrm{R_V}(p, L_r) \cap U) dp. \end{aligned} \tag{9}$$

Unlike (1) (which can only be solved numerically), the integral in (9) can be solved as (see the Appendix):

$$\begin{aligned} &E[Vol(\Theta(q,r) \cap U)] \\ &\approx \begin{cases} \left(L_r - \frac{L_r^2}{4}\right)^d = \left(C_v \cdot r - \frac{C_V^2 \cdot r^2}{4}\right)^d, & L_r < 2 \\ 1, & otherwise, \end{cases} \end{aligned} \tag{10}$$

where C_V is the vicinity constant defined in (8). As mentioned earlier, D_k can be obtained by solving r from $E[Vol(\Theta(q,r) \cap U)] = k/N$, resulting in:

$$D_K \approx \frac{2}{C_V}\left[1 - \sqrt{1 - \left(\frac{k}{N}\right)^{\frac{1}{d}}}\right]. \tag{11}$$

As demonstrated in the experiments, the above formula gives a fairly accurate estimation for a wide range of dimensionalities. To understand this, consider Fig. 4a, which shows a 2D circle and its vicinity rectangle. The part of the circle not covered by the rectangle is partitioned into four pieces, which we call *residue arcs*. Similarly, the rectangle also has four *residue corners* that fall out of the circle. It is important to note that each residue arc and corner have the same area (recall that the area of the circle is the same as that of the rectangle).

Obviously, replacing circles with their vicinity rectangles incurs no error (in computing (9)) if the circle (e.g., C_2 in Fig. 4b, with R_2 as its vicinity rectangle) is entirely contained in the data space U (i.e., $Vol(C_2 \cap U) = Vol(R_2 \cap U)$). For circles (e.g., C_1, with vicinity rectangle R_1) intersecting the boundaries of U, the error is usually limited since the area of the residue arcs inside U cancels (to a certain degree) that of the residue corners inside U (i.e., $Vol(C_2 \cap U) \approx Vol(R_2 \cap U)$). The concepts of residue arcs/corners also extend to higher dimensionalities $d \geq 3$, but the "canceling effects" become weaker as d grows, rendering (11) increasingly erroneous. As shown in Section 5, however, for $d \leq 10$ this equation is accurate enough for query optimization.

3.2 Estimating the Number of Node Accesses

The performance (node accesses) of general queries on hierarchical structures can be described as:

$$NA = \sum_{i=0}^{h-1} (n_i \cdot P_{NAi}), \tag{12}$$

where h refers to the height of the tree (leaf nodes are at level 0), P_{NAi} is the probability that a node at level i is accessed, and n_i is the total number of nodes at level i (i.e., $P_{NAi} \cdot n_i$ is the expected number of node accesses at the ith level). In particular, h and n_i can be estimated as $h = 1 + \lceil log_f(N/f) \rceil$ and $n_i = N/f^{i+1}$, respectively, where N is the cardinality of the data set, b the maximum capacity of a node, and f the node fanout (the average number of entries in a node, typically $f = 69\% \cdot b$ [29]).

In order to estimate P_{NAi} for kNN, we need the average extent s_i along each dimension for a node at level i. If $2^d > N/f = n_0$ (high dimensionality), each leaf node can split only in $d' = \lceil log_2(n_0) \rceil$ dimensions. For these d' dimensions, the extents of a leaf node are 1/2 (i.e., each split is performed at the middle of the dimension), while, for the remaining $(d - d')$ dimensions, the extents are 1. If $2_d \leq n_0$ (low dimensionality), a leaf node splits (possibly more than once) on all dimensions. Further, for uniform data, the data characteristics are the same throughout the data space and, hence [3], 1) all nodes at the same level have similar extents,

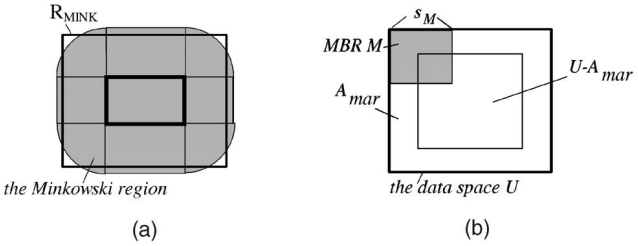


Fig. 5. Approximation of Minkowski region and area that may contain MBR centroids. (a) Rectangular approximation of Minkowski region. (b) The integral area for an MBR.

and 2) each node has identical extent on each dimension. Motivated by this, Boehm [3] provides the following estimation for s_i:

$$S_i = \left(1 - \frac{1}{f}\right)\left(\min\left(\frac{f^{i+1}}{N}, 1\right)\right)^{1/d} \quad (0 \le i \le h-1). \quad (13)$$

Next, we discuss the probability that a MBR M intersects $\Theta(q, D_k)$ or, equivalently, the Minkowski region $\Xi(M, D_k)$ of M (review Fig. 3) contains the query point q. Recall that, according to [6], this probability (6) requires expensive numerical evaluation. To avoid this, we approximate $\Xi(M, D_k)$ with a (hyper) rectangle $\mathrm{R_{MINK}}(M, L_D)$ 1) whose centroid is the same as that of M and 2) whose extent L_D on each dimension is such that $Vol(\mathrm{R_{MINK}}(M, L_D))$ equals $Vol(\Xi(M, D_k))$. Specifically, the volume of $\Xi(M, D_k)$ is calculated as follows [6]:

$$Vol(\Xi(M, D_k)) = \sum_{i=0}^{d}\left(\binom{d}{i} \cdot {s_M}^{d-i} \cdot \frac{\sqrt{\pi^i}}{\Gamma(i/2+1)} D_k^i\right), \quad (14)$$

where $\Gamma(i/2+1)$ is computed as in (7) and s_M refers to the extent of M on each dimension. Thus, L_D can be obtained by solving ${L_D}^d = Vol(\Xi(M, D_k))$, which leads to:

$$L_D = \left[\sum_{i=0}^{d}\left(\binom{d}{i} \cdot {s_M}^{d-i} \cdot \frac{\sqrt{\pi^i}}{\Gamma(i/2+1)} D_k^i\right)\right]^{\frac{1}{d}}. \quad (15)$$

Fig. 5a illustrates $\mathrm{R_{MINK}}(M, L_D)$ together with the corresponding $\Xi(M, D_k)$. Then, the probability P_{NAi} that a node at level i is visited during a kNN query, equals $E[Vol(\Xi(M, D_k) \cap U)]$, i.e., the expected volume between the intersection of $\Xi(M, D_k)$ and the data space U. Replacing $\Xi(M, D_k)$ with $\mathrm{R_{MINK}}(M, L_D)$, we represent P_{NAi} as:

$$\begin{aligned} P_{NAi} &= E[Vol(\Xi(M, D_k) \cap U)] \\ &\approx \int_{U - A_{mar}} Vol(\mathrm{R_{MINK}}(M, L_D) \cap U) dM_c, \end{aligned} \quad (16)$$

where M_c denotes the centroid of M, and A_{mar} is the margin area close to the boundary of U that cannot contain the centroid of any MBR M with extents s_M along each dimension (Fig. 5b shows an example for the 2D case), taking into account the fact that each MBR must lie completely inside the data space. The above integral is solved as (see the Appendix):

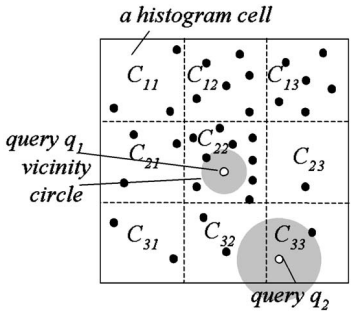


Fig. 6. Histogram example.

$$P_{NAi} \approx \begin{cases} \left(\frac{L_i - (L_i/2 + s_i/2^2}{1 - s_i}\right)^d & \text{if } L_i + s_i < 2 \\ 1 & \text{otherwise} \end{cases} \quad (0 \le i \le h-1), \quad (17)$$

where L_i is obtained from (15) by substituting s_M with s_i. As evaluated in the experiments, (17), albeit derived from approximation, is accurate for dimensionalities $d \le 10$ due to reasons similar to those discussed in Section 3.1 (on the approximation error of vicinity rectangle). Specifically, if the original Minkowski region $\Xi(M, D_k)$ completely falls in the data space, our approximation incurs no error (in the computation of (16)) since

$$Vol(\Xi(M, D_k)) = Vol(\mathrm{R_{MINK}}(M, L_D));$$

otherwise, the error is usually limited due to the canceling effects (note that the concepts of residue arcs/corners introduced in Section 3.1 can be formulated accordingly here).

Combining (7) to (17), we summarize the number of node accesses for kNN queries as follows:

$$NA(k) = \sum_{i=0}^{\log_f \frac{N}{f}}\left[\frac{N}{f^{i+1}} \cdot \left(\frac{L_i - (L_i/2 + s_i/2)^2}{1 - s_i}\right)^d\right], \quad (18)$$

where f is the average node fanout, and s_i the extent of a level-i node given by (13).

3.3 Dealing with Nonuniform Data

The above analysis assumes that the data distribution is uniform. In this section, we extend our results to arbitrary distribution with the aid of histograms. Our technique is the first in the literature to predict the costs of individual queries (see Section 2 for a discussion of the existing analysis). The rationale behind our approach is that data within a sufficiently small region can be regarded as uniform, even though the global distribution may deviate significantly. Thus, the goal is to divide the space into smaller regions and apply the uniform model locally (i.e., within each region). For simplicity, we describe the idea using a regular histogram that partitions the d-dimensional space into H^d cells of equal size (the *histogram resolution* H refers to the number of partitions along each dimension), but our method applies to any histogram with disjoint partitions (such as the Minskew [2]). For each cell c, we maintain the number n_c of data points in it.

Fig. 6 shows a 2D example for a nearest neighbor query q_1 in cell c_{22} (the subscript indicates the second row and

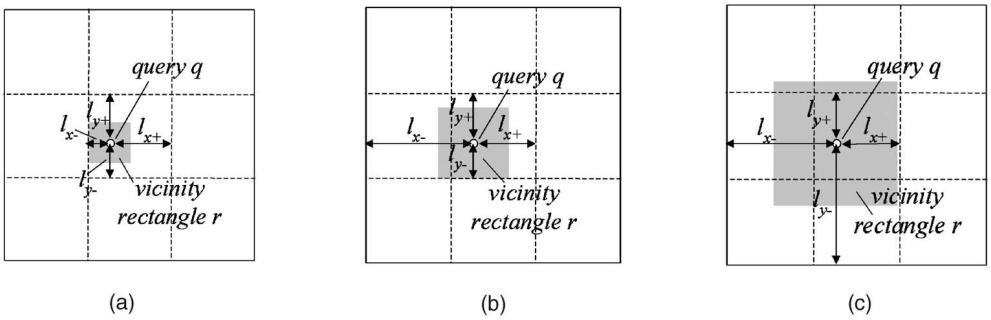


Fig. 7. Estimating L_r. (a) The first iteration. (b) The second iteration. (c) The third iteration.

column) that contains $n_{22} = 9$ points. Since the nearest neighbor of q_1 and its vicinity circle fall completely in c_{22}, in order to estimate the cost we only need to focus on those nodes whose MBRs are in c_{22}. To apply the uniform model directly, we assume that all cells contain the same number of points as c_{22} and estimate the query cost with (18) by simply replacing N with $n_{22} \times H_2$.

The problem is more complex when the vicinity circle does not fall completely in a cell, which happens when 1) the query needs to retrieve a large number of nearest neighbors (e.g., find 10 nearest neighbors for q_1), or 2) the query falls in a sparse cell (e.g., the vicinity circle of query q_2 intersects c_{32} and c_{33}). In this case, we find all the cells intersecting the vicinity circle and use the average density of these cells to synthesize the corresponding uniform data set (after which we directly apply (18)). In Fig. 6, for example, since there are in total three points in c_{32} and c_{33} (i.e., average density 1.5 points per cell), the equivalent uniform data set has $1.5 \times 9 \approx 14$ points. Estimating the size of the vicinity circle, however, is not trivial due to the changes of data density in adjacent cells. To see this, observe that the intersection of a cell with a circle has irregular shape (especially in high-dimensional space), whose computation is difficult and expensive. In the sequel, we discuss a solution that addresses these problems with approximation.

For a kNN query q, we estimate D_k by enlarging the search region progressively until the expected number of points in it equals k. Let $\mathrm{R_V}(q, L_r)$ be the vicinity rectangle centered at q with extent L_r along each dimension. The algorithm increases L_r until it is expected to contain k points, after which D_k can be obtained by dividing L_r with the vicinity constant C_V (as shown in (8), $L_r = C_V \cdot D_k$). Specifically, if c is the cell that contains q, the algorithm initializes a heap HP that contains the distances from q to the boundaries of c (for d-dimensional space, the heap contains $2d$ values). Consider, for example, Fig. 7a, where q falls in cell c_{22}. The content of HP is (in ascending order) $\{l_{x-}, l_{y-}, l_{x+}, l_{y+}\}$ ($x-$ means negative direction of the x-axis, etc). At each iteration, the algorithm removes the minimum value l from HP and enlarges the vicinity rectangle to $L = 2l$. The shaded region in Fig. 7a shows the first vicinity rectangle R obtained from l_{x-}. In order to estimate the number of points falling in R, assume that c_{22} contains n_{22} points; then, the expected number En of points in R is $n_{22} \cdot \mathrm{area}(R)/\mathrm{area}(c_2 2)$, where the areas of R and c_{22} are L^2 and $1/H^2$, respectively. If $k < En$, the vicinity rectangle is too large (it contains more than k points), in which case, L_r is set to $L?(k/En)^{1/d}$ so that the resulting rectangle (with length L_r) contains k points and the algorithm terminates.

If $k > En$, the vicinity rectangle needs to be enlarged further. The algorithm will modify l_{x-} to the distance that q must travel (along direction $x-$) in order to reach the next cell boundary (Fig. 7b) and reinsert it into HP. Before starting the second iteration, the current L and En are preserved in L_{old} and En_{old}, respectively. Similarly to the previous pass, we remove the minimum value l in HP (i.e., l_{y-}), enlarge the vicinity rectangle R to $L = 2l$ (the shaded area in Fig. 7b), and compute the expected number En of data points in R. The vicinity rectangle R now spans two cells, c_{21} and c_{22} (with n_{21} and n_{22} points, respectively); thus, En must be computed according to the properties of both cells:

$$En = n_{21} \cdot \frac{area(R \cap c_{21})}{H^{-d}} + n_{22} \cdot \frac{area(r \cap c_{22})}{H^{-d}}. \tag{19}$$

If the vicinity rectangle contains more than the required number of points ($k < En$), the actual size L_r of the vicinity rectangle is smaller than L (i.e., the current size of R) but larger than L_{old} (the computed size during the previous iteration). To estimate L_r, we interpolate L and L_{old} based on:

$$\frac{{L_r}^d - {L_{old}}^d}{k - En_{old}} = \frac{L^d - {L_r}^d}{En - k}. \tag{20}$$

The reasoning of the above equation is that the number of points falling in a rectangle is linear to its volume. Solving the equation, we have:

$$L_r = \left(\frac{{L_{old}}^d (k - En) - L^d (k - En_{old})}{En_{old} - En} \right)^{1/d}. \tag{21}$$

If k is still larger than En, the algorithm will perform another iteration. Fig. 7c shows the updated l_{y-} and the new vicinity rectangle R, which now intersects six cells. Note that, in general, the set of cells whose extents intersect R can be found efficiently. Since the side length of each cell is $1/H$, we can determine (by applying a simple hash function) the cells that contain the corner points of R. Then, those cells between the corner cells are the ones intersecting R. The algorithm will always terminate because the vicinity rectangle eventually covers the entire data space, at which point the maximum number of neighbors are returned.

```
Algorithm Estimate_D_k (q)
/* this algorithm estimates D_k for a k-NN query q based on an input histogram */
1.  initialize a min-heap HP
2.  compute the cell c that contains q
3.  for each dimension i=1 to d
4.      l_i-=the distance q needs to travel along the negative direction to reach the boundary of c
5.      l_i+=the distance q needs to travel along the positive direction to reach the boundary of c
6.      insert l_i- and l_i+ into HP
7.  end for
8.  En_old=0; l_old=0;
9.  do {
10.     l=deheap HP and set L=2·l
11.     compute the expected number En of points in the vicinity rectangle
12.     if (k≤u)
13.         compute L_r by equation (3-15)
14.         D_k=L_r / C_V and return /* algorithm terminates */
15.     else
16.         L_old=L; En_old=En;
17.         update l and re-insert it into HP
18.     end if
19. }while (true) // algorithm terminates at line 14 eventually
End Estimate_D_K (q)
```

Fig. 8. Algorithm for computing D_k.

Fig. 8 summarizes the pseudocode for general d-dimensional spaces. It is worth mentioning that the application of this approach directly to circular regions would lead to expensive evaluation time due to the high cost of computing the intersection between a circle and a rectangle.

As mentioned earlier, after obtaining D_k, we apply the uniform cost model to estimate the number of node accesses. Specifically, assuming that the average density of the cells intersecting the vicinity rectangle is D (points per cell), then the conceived data set consists of $D \cdot H^2$ points. Hence, (18) produces the estimates for the nonuniform data set by setting N to $D \cdot H^2$. Note that, by doing so, we make an implicit assumption that the data density in the search region does not change significantly. Fortunately, this is the case for many real-world distributions (similar ideas have been deployed in spatial selectivity estimation [29], [2]).

If the histogram resolution H is fixed, the number of cells (and, hence, the histogram size) increases exponentially with the dimensionality. In our implementation, we gradually decrease H as the dimensionality increases (see the experimental section for details), but, in practice, the histogram can be compressed. An observation is that, starting from the medium dimensionality (e.g., 5), many cells contain no data points and, thus, are associated with the same number "0." In this case, adjacent empty cells can be grouped together and a single "0" is stored for the region they represent. Alternatively, [21] and [20] propose more sophisticated compression methods based on wavelet and DCT transformations, respectively; their methods can be directly applied in our case.

4 Query Optimization

Equation (18) is a closed formula that estimates the performance of kNN queries on data partitioning access methods. As we will demonstrate experimentally, it is accurate and efficient in terms of computation cost; thus, it is directly applicable to query optimization. In this section, we present three important optimization heuristics that are made possible by the model. It suffices to discuss uniform distribution because, as shown earlier, the result can be extended to nonuniform data using histograms.

4.1 Sequential Scan versus Best-First Algorithm

The performance of kNN search is affected by the dimensionality d and the number k of neighbors retrieved. It has been shown in [31] that, for single nearest neighbor retrieval $(k = 1)$, sequential scan is more efficient than the best-first (BF) algorithm (see Section 2) after d exceeds a certain threshold. In the sequel, we analyze, for a fixed dimensionality d, the value of K_S such that sequential scan outperforms BF for kNN queries with $k > K_S$ (even for low and medium dimensionality).

We consider that each node in the index corresponds to a single disk page (the general case, where each node occupies multiple pages, can be reduced to this case by enlarging the node size accordingly). To derive K_S, we compare the cost of kNN queries as predicted in (18) with that of sequential scan. In practice, a query optimizer will choose an alternative plan only if the expected number of pages visited by this plan is below a percentage ξ (e.g., 10 percent) of that of a sequential scan (because sequential accesses are significantly cheaper than random ones). For simplicity, we consider that 1) the cost of BF, estimated by (14), includes only the leaf node accesses (i.e., $i = 0$ in the summation) since they constitute the major part of the total cost [3], and 2) sequential scan visits as many disk pages as the number of leaf nodes in the index (in practice, the cost may be higher if each point in the data file contains more information than its leaf entry in the index). Then, sequential scan is expected to outperform BF when the following condition holds:

$$\frac{N}{f} \cdot \left(\frac{L_0 - (L_0/2 + s_0/2)^2}{1 - s_0} \right)^d \geq \frac{N}{f} \cdot \xi. \tag{22}$$

Recall that L_0, computed using (15) (replacing s_M with s_0, i.e., the leaf node extent estimated as in (13)), depends on D_k (i.e., the distance from the query point to its k-nearest neighbor) which in turn is a function of k. K_S is the smallest value of k that makes the two sides of the inequality equal. The resulting equation can be written as:

$$-\frac{{L_0}^2}{4} + \left(1 - \frac{s_0}{2}\right) L_0 - \left(\frac{{s_0}^2}{4} + \xi^{\frac{1}{d}} (1 - s_0) \right) = 0. \tag{23}$$

If L_0 is the root that satisfies the above equation, then K_S is derived as (from (11) and (15)):

$$K_s = N \cdot \left[1 - \left(1 - \frac{C_v}{2C_{MINK}} (L_0 - s_0) \right)^2 \right]^d. \tag{24}$$

4.2 Optimizing the Node Size

An index node can contain a number B of consecutive disk pages. The cost of each node access can be modeled as $(T_{SEEK} + T_{IO} \cdot B)$, where T_{SEEK} is the seek time for a disk operation, and T_{IO} is the time for transmitting the data of a single page. In practice, T_{SEEK} is significantly larger than T_{IO} (typically, $T_{SEEK} = 10\text{ms}$ and $T_{IO} = 1\text{ms}$ per 1k bytes). Let $NA(B, k)$ be the number of node accesses for a kNN query when each index node occupies B pages. Thus, the total query cost T_{TOTAL} is $(T_{SEEK} + T_{IO} \cdot B) \cdot NA(B, k)$. Note that a higher value for B decreases the number of nodes in the index (leading to larger fanout f in (18)) and, hence, reduces $NA(B, k)$. On the other hand, a higher B may increase the data transfer time $T_{IO} \cdot B \cdot NA(B, k)$ (particularly, if B is large, then the index structure degenerates into a sequential file). This fact has been utilized in [9] to dynamically adjust the index node size and optimize the performance under the L_∞ distance metric. Instead, in the sequel, we aim at quantifying, for the Euclidean distance metric, the optimal value of B that minimizes the total cost of retrieving k nearest neighbors.[5] For this purpose, we rewrite the total cost T_{TOTAL} as (only counting the leaf accesses):

$$\begin{aligned} T_{TOTAL}(B) = (T_{SEEK} + T_{IO} \cdot B) \cdot \frac{N}{f(B)} \\ \cdot \left(\frac{L(B) - (L(B)/2 + s(B)/2)^2}{1 - s(B)} \right)^2, \end{aligned} \tag{25}$$

where $L(B)$, $f(B)$, and $s(B)$ denote the values of L, f, and s as functions of B. Specifically, $f(B) = 0.69B \cdot B_{size}/O_{size}$ (B_{size} and O_{size} denote the sizes of a disk page and an object entry, respectively), and $s(B)$ and $L(B)$ are obtained from (13) and (15). Optimizing B (i.e., finding the minimum T_{TOTAL}) requires solving the derivative:

5. Berchtold et al. [7] also evaluate the optimal index node size in terms of the number of disk pages. Their discussion, however, is for single nearest neighbor queries in high-dimensional spaces. Further, they assume the query points lie at the diagonal of the universe. Our analysis does not have these constraints.

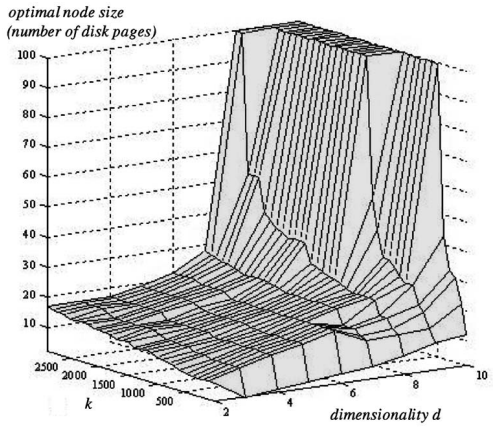


Fig. 9. Optimal node sizes (disk page = 1k bytes).

$$\frac{dT_{TOTAL}(B)}{dB} = 0. \tag{26}$$

Fig. 9 demonstrates the solutions of (26) by plotting the optimal B as a function of k and d (where the maximum B is set to 100) for a data set of 100K points (disk page size set to 1k bytes). For all dimensionalities, the optimal node size increases with k. This is because a kNN query with larger k needs to access more pages with data points around the query point. Therefore, larger B reduces the node visits, which also leads to fewer random accesses. Furthermore, notice that the growth of B is faster as the dimensionality increases. In particular, for $d = 10$ and $k > 500$, the optimal B is larger than 100 blocks, i.e., the index structure degenerates into a sequential file.

4.3 Optimizing Incremental kNN Queries

For incremental kNN queries (distance browsing) the number k of neighbors to be retrieved is unknown in advance. In the worst case, k may equal the cardinality N of the data file, which is also the case for *distance sorting* (i.e., output all data points ranked according to their distance from a query point). Recall that, in order to answer such queries, the BF algorithm (reviewed in Section 2) must store in a heap all entries of the nodes visited so far. As noted in [7], if the final number of neighbors retrieved is large, the size of the heap may exceed the main memory and part of it needs to be migrated to the disk. This causes disk thrashing (i.e., frequent information exchange between the main memory and disk) and compromises the query performance significantly.[6] To explain this qualitatively, we estimate the heap size when k neighbors have been reported, by combining the following facts: 1) On average, f (i.e., the node fanout) entries are inserted to the heap when a node is visited (hence, if $NA(k)$ nodes are visited to find k neighbors, $f \cdot NA(k)$ entries are inserted), 2) a leaf entry (i.e., a point) is removed from the heap when the point is reported, and 3) a nonleaf entry is removed when its node is visited. Therefore, the heap contains $f \cdot NA(k) - k - (NA(k) - 1) = (f - 1) \cdot NA(k) - k + 1$ entries after reporting k nearest neighbors. Thus, the heap size may keep growing

6. The problem is not as severe for nonincremental kNN because the distance from the query point to the currently found kth nearest neighbor can be used to reduce the heap size (see [18] for details).

until most of the nodes in the index have been visited (this is experimentally confirmed in the next section).

Based on our model, we present a multipass distance browsing algorithm (using the query example "find the 10 nearest cities of New York with populations larger than one million"). Given the available memory size M, the algorithm first estimates the largest number k_1 of neighbors to be found such that the expected size of the heap will not exceed M. Then, it performs the first pass, i.e., an ordinary k_1-NN query using BF. If 10 cities with satisfactory populations are found during this pass (possibly before k_1 cities are examined), the algorithm terminates. Otherwise, a second pass is executed to retrieve the next k_2 neighbors, where k_2 is an estimated number such that the heap of the second pass will not exceed the memory size M. This process is repeated until 10 cities satisfying the population condition are encountered. The second and subsequent passes are performed in the same way as the first pass except that they include an additional pruning heuristic: Let the current pass be the ith one ($i \geq 2$); then, if the maximum (actual) distance of a nonleaf (leaf) entry is smaller than the distance of the farthest neighbor found in the $(i-1)$th pass, this entry is immediately discarded (it has been considered by a previous pass). Further, our BF algorithm has the following difference from the one introduced in Section 2: The leaf and nonleaf entries are stored in separate heaps, called the *nonleaf heap* and *leaf heap* (i.e., with size k_i at pass i), respectively.

Now, it remains to estimate k_i subject to M, which is measured in terms of the number of heap entries. If E_{NON} is the number of nonleaf entries that would simultaneously reside in the nonleaf heap, then k_i should satisfy the property: $E_{NON} + k_i \leq M$. Next, we estimate an upper bound for E_{NON}. Let $NonA_i$ be the number of nonleaf nodes accessed during the ith pass; then, $E_{NON} \leq f \cdot NonA_i - NonA_i$, where $f \cdot NonA_i(NonA_i)$ is the total number of nonleaf entries inserted into (removed from) the heap in the ith pass. Observe that in the worst case, the ith pass would have to access K_i neighbors, where $K_i = \sum_{m=1\sim i} k_m$. (i.e., all neighbors reported in previous passes must be visited). Hence, $NonA_i \leq NA_{NON}(K_i)$, where $NA_{NON}(K_i)$ is the number of nonleaf node accesses in reporting K_i neighbors, which is obtained from (18) as follows (note the summation below starts from level 1):

$$NA_{NON}(K_i) = \sum_{i=1}^{\log_f \frac{N}{f}} \left[\frac{N}{f^{i+1}} \cdot \left(\frac{L_i - (L_i/2 + s_i/2)^2}{1 - s_i} \right)^d \right]. \quad (27)$$

Therefore, k_i can be obtained by solving the following equation:

$$f \cdot NA_{NON}(K_i) - NA_{NON}(K_i) + k_i = M. \quad (28)$$

5 Experimental Evaluation

This section experimentally evaluates the proposed model and optimization techniques, using the R*-tree [4] as the underlying spatial access method. We deploy 1) uniform data sets that contain 100k points in 2 to 10-dimensional data spaces (where each axis is normalized to have unit length) and 2) real data sets *Color* and *Texture* (both available at the *UCI KDD archive* [30]), containing 68k 4D and 8D points, respectively, that describe features in the color and texture histograms of images in the Corel collection. Unless otherwise stated, the node size equals the disk page size (set to 1k bytes) such that node capacities (i.e., the maximum number of entries in a node) range from 10 (for 10 dimensions) to 48 (for two dimensions). We select a relatively small page size to simulate practical situations where the database is expected to be considerably larger. All the experiments are performed using a Pentium III 1GHz CPU with 256 mega bytes memory. Section 5.1 first examines the precision of our formulae for estimating the nearest distance (i.e., the distance from a query point to its kth nearest neighbor) and query cost. Section 5.2 demonstrates the efficiency of the query optimization methods.

5.1 Evaluation of Nearest Distance and Query Cost Prediction

For all the experiments in this section, we use workloads each containing 100 query points that uniformly distribute in the data space and retrieve the same number k of neighbors. Starting from uniform data, the first experiment examines the accuracy of estimating the nearest distance D_k. For this purpose, we measure the average, minimum, and maximum D_k of all the queries in a workload and compare them with the corresponding estimated value (since we apply (11) directly, without histograms, there is a single estimation for all queries in the same workload). Fig. 10a plots the nearest distance as a function of dimensionality for $k = 1, 500$ (each vertical line segment indicates the range of the actual D_k in a workload), and Fig. 10b illustrates the results as a function of k for dimensionality 5. We evaluate our model up to a relatively high value of k because the boundary effect is not significant for small k (for the dimensionalities tested). Observe that the variance in actual D_k is larger as the dimensionality or k increases[7] since the boundary effect is more significant. The estimated values capture the actual ones very well in all cases.

For comparison, we also implemented the model of [3] which does not use approximation but involves complex integrals. As expected, the estimated values produced by this model are even closer to the actual (average) ones, but at the expense of high evaluation overhead (the time to produce an estimate ranges from 0.8 seconds, for $d = 2$, to 5 seconds, for $d = 10$). This cost can be reduced by precomputing a set of values as discussed in [3], which, in our implementation, results in computed values with total size 2M bytes for the same estimation accuracy. Having considerable size, these values may need to be stored on the disk and incur disk accesses. On the other hand, the running time of our model (11) is negligible (not even measurable) and the space overhead is zero.

Next, we evaluate (18) that predicts the query cost (in terms of the number of node accesses) for uniform data. In Fig. 11a, we fix k to 1,500 and illustrate the actual and estimated query costs as a function of dimensionality. For

7. The variance is also large when $k = 1$ (i.e., single nearest neighbor retrieval), in which case, the nearest distance can be arbitrarily close to 0 as the query point approaches a data point.

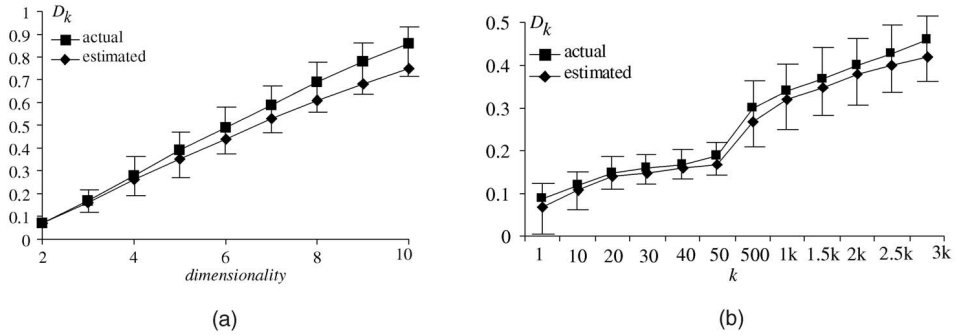


Fig. 10. Evaluation of D_k estimation (uniform). (a) D_k versus d ($k = 1,500$). (b) D_k versus k ($d = 5$).

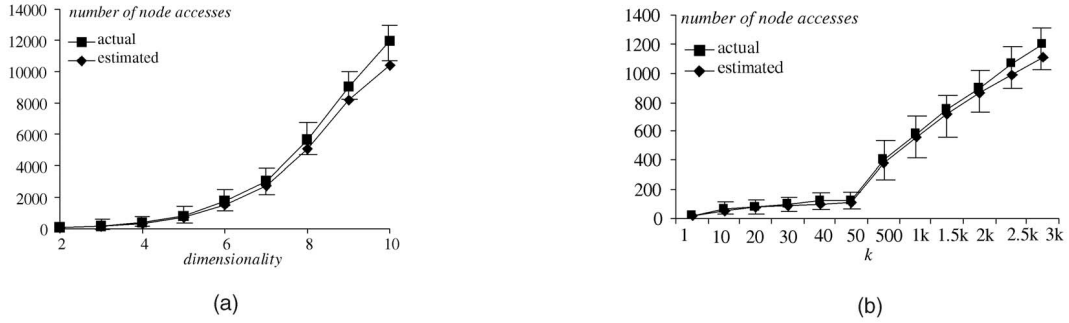


Fig. 11. Evaluation of query cost estimation (uniform). (a) Query cost versus d ($k = 1,500$). (b) Query cost versus k ($d = 5$).

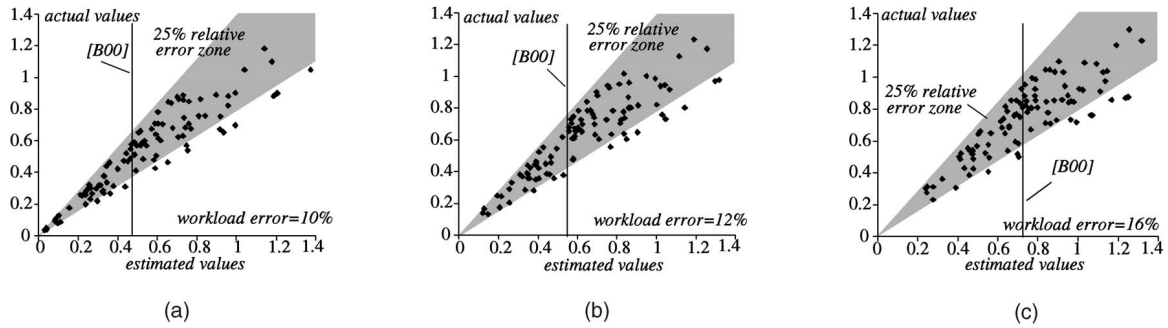


Fig. 12. D_k evaluation (*Color* data set). (a) $k = 10$. (b) $k = 500$. (c) $k = 3,000$.

the actual cost, we report the average value, as well as the cost range (using a vertical line segment), of each workload. As expected, the search overhead increases exponentially with the dimensionality, confirming the findings of [31]. In Fig. 11b, we fix $d = 5$ and study the effect of various values of k (from 1 to 3,000). Once again the estimated values are fairly accurate, and (similarly to Fig. 10) the precision is relatively lower for large d and k where the boundary effect (and, hence, the variance of different queries' overhead) is more significant. As with D_k estimation, our model (18) produces an estimate almost instantly, while the model of [3] takes from 2 seconds (for $d = 2$) to 40 seconds (for $d = 10$) without precomputation. Note that the precomputation method in [3] is not applicable here because, as explained in Section 2, it applies only to the high-dimensional space where node extents are either $1/2$ or 1, and its extension to lower-dimensional spaces is not straightforward.

Having demonstrated the effectiveness of the model for uniform distributions, we proceed to evaluate the histogram technique proposed in Section 3.3 for nonuniform data. We use a regular-grid histogram with resolution H (i.e., the number of partitions along each dimension), which is decided according to the available memory. Specifically, the histogram size is limited to 200k bytes (or, equivalently, to 50K cells) and H is set to $\lfloor 50,000^{1/d} \rfloor$ for the d-dimensional space (we do not apply any compression method). Particularly, for the 4D data set *Color*, $H = 15$, while, for the 8D data set *Texture*, $H = 4$. Queries in the same workload uniformly distribute in the data space and return the same number k of neighbors. Let act_i and est_i denote the actual and estimated values for the ith query $(1 \leq i \leq 100)$; then, the workload error equals $(1/100) \cdot \sum_i |act_i - est_i|/act_i$.

Fig. 12a shows, for data set *Color*, the actual and estimated D_k of each query (i.e., the horizontal and vertical coordinates of a plotted point, respectively) in the workload with $k = 10$. Ideally, all points would fall on the diagonal of the *act-est* space (i.e., actual and estimated values are equal). The shaded area covers queries for which our

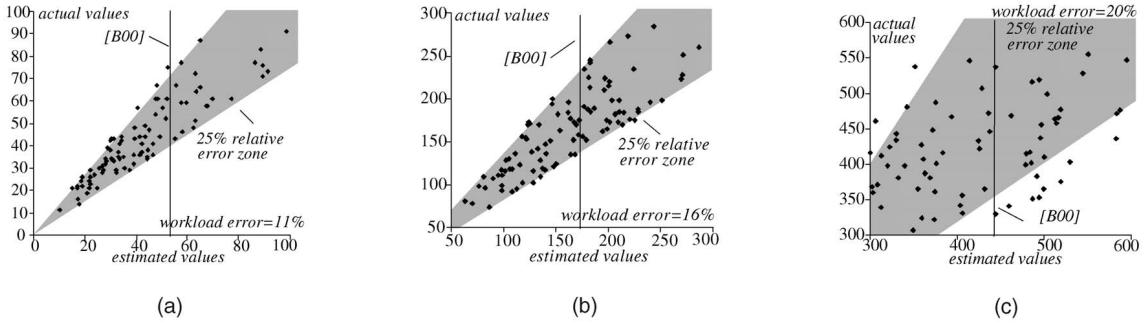


Fig. 13. Query cost (number of node accesses) evaluation (*Color* data set). (a) $k = 10$. (b) $k = 500$. (c) $k = 3,000$.

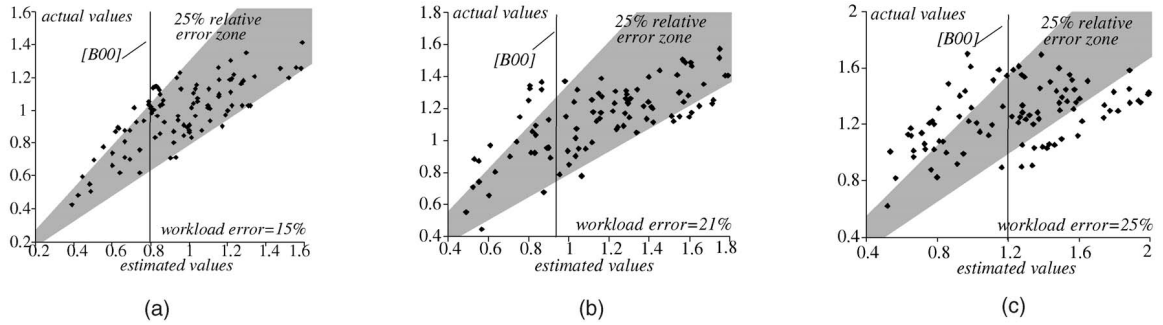


Fig. 14. D_k evaluation (*Texture* data set). (a) $k = 10$. (b) $k = 500$. (c) $k = 3,000$.

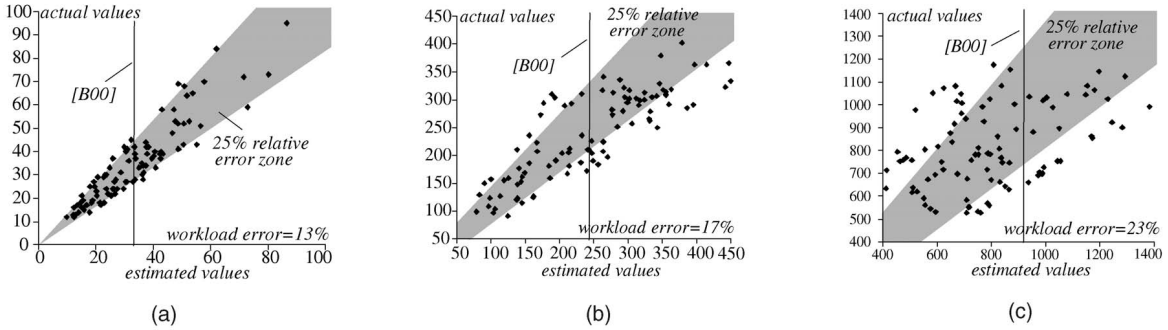


Fig. 15. Query cost (number of node accesses) evaluation (*Texture* data set). (a) $k = 10$. (b) $k = 500$. (c) $k = 3,000$.

technique yields up to 25 percent relative error. For comparison, we also include the estimate of [3], which, as introduced in Section 2, provides a single estimate (i.e., 0.45, represented as a vertical line) based on the data set's fractal dimension. The estimate of [3] is clearly inadequate because it does not capture the actual performance at all (i.e., various queries have give very different results). Particularly, notice that this estimate is smaller than most actual D_k because 1) [3] assumes the query distribution follows that of data, while, in our cases, queries can appear at any location of the data space, and 2) the nearest distances of queries in sparse areas (where there are few data points) are longer than those in data-dense areas. On the other hand, our estimation method is able to provide accurate prediction for most queries. Specifically, 90 percent of the queries have less than 25 percent relative error, and the workload error is 10 percent (as indicated in the figure). Similar observations can be made for Figs. 12b and 12c ($k = 500$ and 3,000, respectively), as well as Fig. 13 that evaluates the accuracy of cost estimation for various values of k. Fig. 14 and Fig. 15 demonstrate similar experiments for *Texture*. Note that, by comparing the diagrams with Fig. 12 and Fig. 13, the error is higher because 1) the boundary effect is more serious as the dimensionality increases (in which case, approximating a circle with a hyperrectangle tends to be more erroneous), and 2) given the same size limit for the histogram, its resolution drops considerably (i.e., 4 for *Texture*) so that the uniformity in each cell degrades.

5.2 Evaluation of Query Optimization Methods

In this section, we evaluate the effectiveness of the query optimization techniques presented in Section 4 using uniform data sets. As discussed in Section 4.1, for a kNN query, sequential scan outperforms the best-first (BF) algorithm if k exceeds a certain value of K_S (given in (24)). The next experiment evaluates K_S with respect to different percentage thresholds ξ (parameter of (24)). For example, if $\xi = 5$ percent and sequential scan requires 1,000 node accesses, then BF is considered worse if it visits more than 50 nodes.

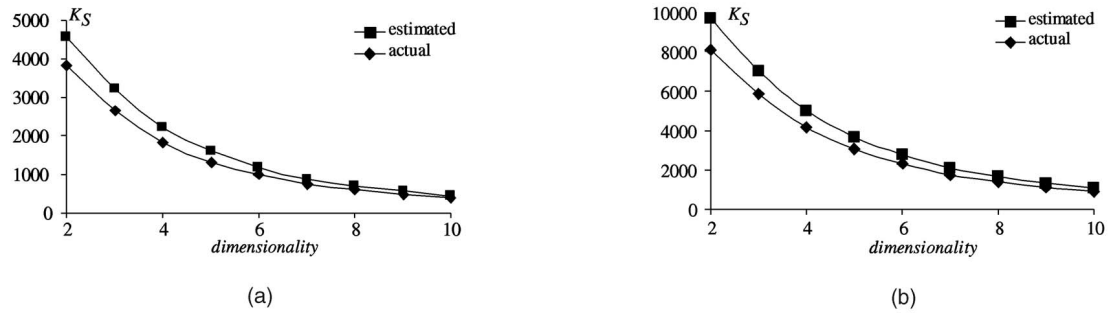


Fig. 16. K_S versus dimensionality (cardinality = 100K). (a) K_S versus d ($\xi = 5$ percent). (b) K_S versus d ($\xi = 10$ percent).

Figs. 16a and 16b compare the estimated and actual K_S when $\xi = 5$ percent and 10 percent, respectively, using a uniform data set with 100K points. In both cases, the actual values of K_S are around 20 percent higher than the corresponding estimated values. Observe (by comparing the two figures) that K_S increases with ξ because higher ξ implies faster seek time and, thus, the overhead of random accesses is lower for index-based approaches. Furthermore, the fact that K_S decreases with dimensionality is consistent with the previous findings [31], [8], [7] that the BF algorithm degenerates into sequential scans for high dimensionalities.

To demonstrate the gains of different node sizes, we created R*-trees whose nodes occupy between 1 and 100 disk pages for 100K uniform points. Fig. 17 shows the performance (in terms of total disk access time according to (26)) of 500-NN queries as a function of node sizes for 2, 5, and 10 dimensions ($T_{SEEK} = 10$ms and $T_{IO} = 1$ms per 1k bytes). It is clear that, for all dimensionalities, the performance initially improves as the node size increases, but deteriorates after the optimal size, meaning that the overhead of data transfer time does not pay off the reduced seek time. Notice that the optimal values (16, 18, 100 for 2, 5, 10 dimensions, respectively) increase with the dimensionality, which is consistent with our predictions in Fig. 9.

Next, we evaluate the memory requirement of the best first (BF) algorithm for retrieving nearest neighbors incrementally. Fig. 18 shows the heap size (in terms of the number of entries) for a distance browsing query located at the center of the 2, 5, and 10-dimensional spaces, respectively (data set cardinality = 100K). The amount of required memory initially increases with k (i.e., the number of neighbors retrieved), but decreases after most nodes of the tree have been accessed. For low dimensions (e.g., 2), the heap size is small enough to fit in memory even for very large k. In case of higher dimensions (e.g., 5 and 10), however, the heap requirements are prohibitive even for moderate values of k. For 10 dimensions, for example, if the available memory can accommodate 30K entries (i.e., 30 percent of the data set), then disk thrashing occurs for k as low as 100.

Finally, we compare the multipass algorithm (MP in short) described in Section 4.3 with the best-first method (BF) in situations where the available memory is not enough for the heap. For this purpose, we use both algorithms to perform distance sorting, which outputs the data points in ascending order of their distances to the center of the data space. In Fig. 19, we fix the dimensionality to 5, and measure the page accesses for various memory sizes accounting for 20 to 100 percent of the maximum heap size shown in Fig. 18. For the implementation of BF, we deploy the three-tier heap management policy of [18]. As expected, when the memory is enough for the entire heap (i.e., the 100 percent case), the two algorithms have identical performance and MP behaves like BF. When the amount of memory is close to 100 percent, BF is slightly better because, in this case, the overhead of performing several passes (for MP) is greater than the disk penalty incurred

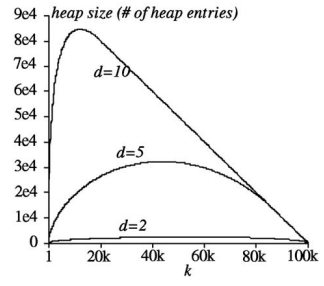


Fig. 18. Heap size for incremental kNN versus k.

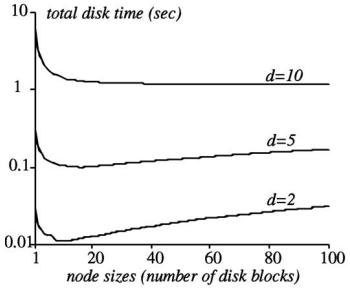


Fig. 17. Total query cost versus node size.

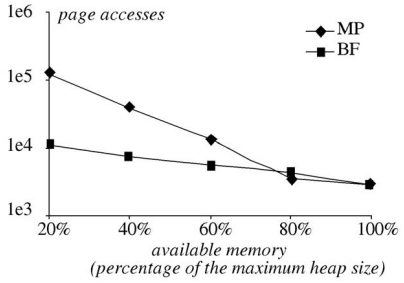


Fig. 19. Query cost versus available memory.

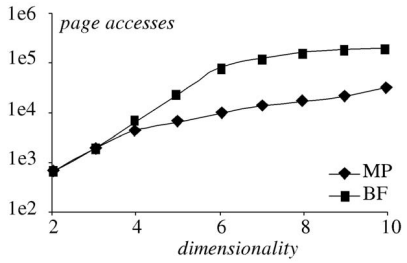


Fig. 20. Query cost versus d.

from disk trashing of BF. In all the other cases, MP is superior and the performance gap increases as the memory decreases. Particularly when the available memory is enough for only 20 percent of the maximum heap size, MP outperforms BF by an order of magnitude.

In Fig. 20, we fix the memory size (to 10 percent of the database size) and evaluate the cost of distance sorting as a function of dimensionality. Notice that the difference of MP and BF is negligible for low dimensionalities (2 and 3), which is expected because, in these cases, the memory is large enough for the corresponding heaps. As the dimensionality increases, the required heap size (for BF) grows exponentially, resulting in severe buffer thrashing. It is interesting that the relative performance of MP and BF stabilizes for $d \geq 6$ (Fig. 20) because, for higher dimensionalities, MP accesses a significant part of the tree at subsequent passes due to the large distances of most data points from the query point (as discussed in [31], the average distance between two points grows exponentially with the dimensionality). On the other hand, BF incurs similar costs after certain dimensionality since it essentially accesses all nodes and inserts all the entries into the heap. As shown in the figure, MP outperforms BF in most cases by an order of magnitude.

6 Conclusion

This paper proposes a cost model for kNN search applicable to a wide range of dimensionalities with minimal computational overhead. Our technique is based on the novel concept of vicinity rectangles and Minkowski rectangles (instead of the traditional vicinity circles and Minkowski regions, respectively), simplifying the resulting equations. We confirm the accuracy of the model through extensive experiments, and demonstrate its applicability by incorporating it in various query optimization problems.

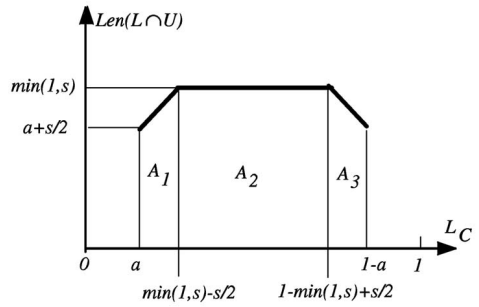


Fig. 22. Change of $Len(L \cap U)$ as a function of L_C.

Compared to previous work, the proposed model has the following advantages: 1) Its small computational cost makes it ideal for real-time query optimization, 2) it permits the application of conventional multidimensional histograms for kNN search, and 3) the derived formulae can be easily implemented in an optimizer and combined with other techniques (e.g., range selectivity estimation [29], [2] for constrained kNN queries [16]).

On the other hand, the model has certain limitations that motivate several directions for future work. First, the approximation scheme yields increasing error with dimensionality, such that, after 10 dimensions, it is no longer suitable for query optimization due to inaccuracy. A possible solution for this problem may be to develop alternative approximation methods or identify some compensation factors to reduce the error. Second, our histogram-based algorithm (for cost estimation on nonuniform data) only supports histograms whose bucket extents are disjoint, which, however, is not satisfied in some high-dimensional histograms (such as the one proposed in [17]). This limits its applicability in these scenarios. Further, another interesting future work is to extend the cost model and related optimization techniques to closest pair queries [11], which retrieve the k closest pairs of objects from two data sets. Finally, the employment of the model for nonpoint data sets is also a subject worth studying.

Appendix

Here, we present a solution to the following problem that is essential for considering boundary effects in cost analysis (generalizing the analysis in [3]). Given a (hyper) rectangle R, 1) whose extents are s along all dimensions and 2) whose centroid R_C is restricted to the region A_{mar} that has margin a (which is an arbitrary constant in the

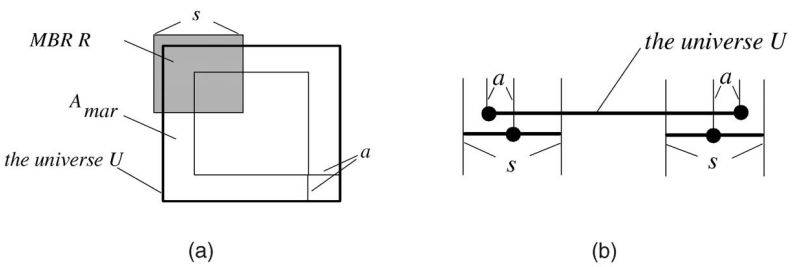


Fig. 21. The integral area. (a) 2D case. (b) 1D version of the problem.

range [0, 0.5]) to each boundary of the (unit) data space U (Fig. 21a shows a 2D example), compute the average area (volume) of $R \cap U$ over all positions of R_C or, more formally, solve the following integral:

$$avgVol(R) = \iint_{U-A_{mar}} Vol(R \cap U) dR_C. \quad (29)$$

We first solve the 1D version of this problem (Fig. 21b). Specifically, given a line segment L of length s whose center L_C must have distance at least a from the end points of the data space (i.e., a unit line segment), solve the integral:

$$avgLen(L) = \int_{[a,1-a]} Len(L \cap U) dL_C, \quad (30)$$

where the integral region $[a, 1-a]$ corresponds to the possible positions of L_C.

The solution is straightforward when 1) $a \geq s/2$ and 2) $a + s/2 \geq 1$. For 1), the line segment is always within the data space; hence, $avgLen(L) = s$. On the other hand, the line segment always covers the entire data space for case 2); thus, $avgLen(L) = 1$. Next, we focus on the case that $2a < a + s/2 < 1$. As L_C gradually moves from position a to $1a$, $Len(L \cap U)$ initially increases (from $a + s/2$) before reaching a maximum value, after which $Len(L \cap U)$ decreases (finally to $a + s/2$). This transition is plotted in Fig. 22.

The average length of $L \cap U$ can be obtained by summing the areas of the regions of the trapezoids A_1, A_2, A_3 and dividing the sum by $1 - 2a$ (i.e., the length of all positions of L_C). Thus, integral (30) can be solved as:

$$avgLen(L) = \frac{area(A_1 + A_2 + A_3)}{1 - 2a} = \frac{s - (s/2 + a)^2}{1 - 2a}. \quad (31)$$

For general d-dimensional spaces, the following equation holds:

$$avgVol(R) = \prod_{i=1}^{d} avgLen(R_i),$$

where R_i is the projection of R along the ith dimension. By (31), the above equation can be solved into the closed form:

$$avgVol(R) = \left[\frac{s - (s/2 + a)^2}{1 - 2a}\right]^d.$$

ACKNOWLEDGMENTS

This work was supported by grants HKUST 6180/03E and HKUST 6197/02E from Hong Kong RGC. The authors would like to thank the anonymous reviewers for their insightful comments.

REFERENCES

[1] S. Arya, D. Mount, and O. Narayan, "Accounting for Boundary Effects in Nearest Neighbor Searching," *Proc. Ann. Symp. Computational Geometry*, 1995.

[2] S. Acharya, V. Poosala, and S. Ramaswamy, "Selectivity Estimation in Spatial Databases," *Proc. ACM SIGMOD Conf.*, 1999.

[3] C. Boehm, "A Cost Model for Query Processing in High Dimensional Data Spaces," *ACM Trans. Database Systems*, vol. 25, no. 2, pp. 129-178, 2000.

[4] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," *Proc. ACM SIGMOD Conf.*, 1990.

[5] S. Berchtold, D. Keim, and H. Kriegel, "The X-Tree: An Index Structure for High-Dimensional Data," *Proc. Very Large Database Conf.*, 1996.

[6] S. Berchtold, C. Boehm, D. Keim, and H. Kriegel, "A Cost Model for Nearest Neighbor Search in High-Dimensional Data Space," *Proc. ACM Symp. Principles of Database Systems*, 1997.

[7] S. Berchtold, C. Boehm, D. Keim, F. Krebs, and H. Kriegel, "On Optimizing Nearest Neighbor Queries in High-Dimensional Data Spaces," *Proc. Int'l Conf. Database Theory*, 2001.

[8] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?" *Proc. Int'l Conf. Database Theory*, 1999.

[9] S. Berchtold and H. Kriegel, "Dynamically Optimizing High-Dimensional Index Structures," *Proc. Int'l Conf. Extending Database Technology*, 2000.

[10] J. Cleary, "Analysis of an Algorithm for Finding Nearest Neighbors in Euclidean Space," *ACM Trans. Math. Software*, vol. 5, no. 2, pp. 183-192, 1979.

[11] A. Corral, Y. Manolopoulos, Y. Theodoridis, and M. Vassilakopoulos, "Closest Pair Queries in Spatial Databases," *Proc. ACM SIGMOD Conf.*, 2000.

[12] P. Ciaccia, M. Patella, and P. Zezula, "A Cost Model for Similarity Queries in Metric Spaces," *Proc. ACM Conf. Principles on Database Systems*, 1998.

[13] J. Friedman, J. Bentley, and R. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209-226, 1977.

[14] C. Faloutsos and I. Kamel, "Beyond Uniformity and Independence, Analysis of R-Trees Using the Concept of Fractal Dimension," *Proc. ACM Conf. Principles of Database Systems*, 1994.

[15] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-Series Databases," *Proc. ACM SIGMOD Conf.*, 1994.

[16] H. Ferhatosmanoglu, I. Stanoi, D. Agarwal, and A. Abbadi, "Constrained Nearest Neighbor Queries," *Proc. Symp. Spatial and Temporal Databases*, 2001.

[17] D. Gunopulos, G. Kollios, V. Tsotras, and C. Domeniconi, "Approximate Multi-Dimensional Aggregate Range Queries over Real Attributes," *Proc. ACM SIGMOD Conf.*, 2000.

[18] G. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *Proc. ACM Trans. Database Systems*, vol. 24, no. 2, pp. 265-318, 1999.

[19] F. Korn, B. Pagel, and C. Faloutsos, "On the 'Dimensionality Curse' and the 'Self-Similarity Blessing'," *IEEE Trans. Knowledge and Database Eng.*, vol. 13, no. 1, pp. 96-111, 2001.

[20] J. Lee, D. Kim, and C. Chung, "Multidimensional Selectivity Estimation Using Compressed Histogram Information," *Proc. ACM SIGMOD Conf.*, 1999.

[21] Y. Matias, J. Vitter, and M. Wang, "Wavelet-Based Histograms for Selectivity Estimation," *Proc. ACM SIGMOD Conf.*, 1998.

[22] B. Pagel, F. Korn, and C. Faloutsos, "Deflating the Dimensionality Curse Using Multiple Fractal Dimensions," *Proc. IEEE Int'l Conf. Database Eng.*, 2000.

[23] A. Papadopoulos and Y. Manolopoulos, "Performance of Nearest Neighbor Queries in R-Trees," *Proc. Int'l Conf. Database Theory*, 1997.

[24] N. Roussopoulos, S. Kelly, and F. Vincent, "Nearest Neighbor Queries," *Proc. ACM SIGMOD Conf.*, 1995.

[25] R. Sproull, "Refinements to Nearest Neighbor Searching in K-Dimensional Trees," *Algorithmica*, pp. 579-589 1991.

[26] T. Seidl and H. Kriegel, "Efficient User-Adaptable Similarity Search in Large Multimedia Databases," *Proc. Conf. Very Large Databases*, 1997.

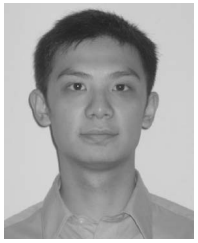
[27] Y. Sakurai, M. Yoshikawa, S. Uemura, and H. Kojima, "The A-Tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation," *Proc. Conf. Very Large Databases*, 2000.

[28] Y. Tao and D. Papadias, "Adaptive Index Structures," *Proc. Conf. Very Large Database*, 2002.

[29] Y. Theodoridis and T. Sellis, "A Model for the Prediction of R-Tree Performance," *Proc. ACM Conf. Principles on Database Systems*, 1996.

[30] UCI KDD archive, http://kdd.ics.uci.edu/, 2002.
[31] R. Weber, H. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces," *Proc. Conf. Very Large Databases,* 1998.

Yufei Tao received the diploma from the South China University of Technology in August 1999 and the PhD degree from the Hong Kong University of Science and Technology in July 2002, both in computer science. After that, he was a visiting scientist at Carnegie Mellon University and is currently an assistant professor in the Department of Computer Science at the City University of Hong Kong. He is also the winner of the Hong Kong Young Scientist Award 2002 from the Hong Kong Institution of Science. His research includes query algorithms and optimization in temporal, spatial, and spatio-temporal databases.



Jun Zhang received his diploma from the South China University of Technology in July 2000, and the PhD degree from the Hong Kong University of Science and Technology in January 2004. He is currently an assistant professor in the Division of Information Systems at the Nanyang Technological University, Singapore. His research interests include indexing techniques and query optimization in spatial and spatio-temporal databases.

Dimitris Papadias is an associate professor in the Department of Computer Science at the Hong Kong University of Science and Technology (HKUST). Before joining HKUST, he worked at various places, including the Data and Knowledge Base Systems Laboratory-National Technical University of Athens (Greece), the Department of Geoinformation-Technical University of Vienna (Austria), the Department of Computer Science and Engineering-University of California at San Diego, the National Center for Geographic Information and Analysis-University of Maine, and the Artificial Intelligence Research Division-German National Research Center for Information Technology (GMD).

Nikos Mamoulis received a diploma in computer engineering and informatics in 1995 from the University of Patras, Greece, and the PhD degree in computer science in 2000 from the Hong Kong University of Science and Technology. Since September 2001, he has been an assistant professor in the Department of Computer Science at the University of Hong Kong. In the past, he has worked as a research and development engineer at the Computer Technology Institute, Patras, Greece, and as a postdoctoral researcher at the Centrum voor Wiskunde en Informatica (CWI), The Netherlands. His research interests include spatial, spatio-temporal, multimedia, object-oriented, and semistructured databases, and constraint satisfaction problems.