To Reveal the Performance Secrets of the Newest NN Searching Algorithm

Yaokai Feng
Graduate School of Information Science and Elec. Eng., Kyushu University, Japan
fengyk@is.kyushu-u.ac.jp

Kunihiko Kaneko
Graduate School of Information Science and Elec. Eng., Kyushu University, Japan
kaneko@is.kyushu-u.ac.jp

Akifumi Makinouchi
Department of Information Network Engineering, Kurume Institute of Technology, Japan
akifumi@cc.kurume-it.ac.jp

Abstract
Nearest Neighbor (NN) search has been widely used in spatial databases and multimedia databases. Incremental NN (INN) search algorithm is regarded as the optimal NN search because of the minimum number of node accesses and it can be used no matter whether the number of objects to be retrieved is fixed or not in advance. This paper presents an analytical model for estimating performance of the INN search algorithm. For the first time, our model takes m (the number of neighbor objects reported finally), n (the cardinality of database) and d (the dimensionality) as parameters, focusing on the number of node accesses (not only the number of accessed leaf nodes) and the length of the priority queue. Using our model, dimensionality curse is mathematically revealed for an arbitrary number of NN objects retrieved. In our model, (1) for the first time, the two key factors of $d_m$ (the distance from the m-th NN object to the query point) and $\sigma_h$ (the side length of each node) are estimated using their upper bounds and their lower bounds, which is helpful to effectiveness of our model, especially in high-dimensional spaces; (2) for the first time, the possible difference of fanouts among the leaf nodes, the root node and the others is taken into account. The theoretical analysis is verified by experiments.

1. Introduction
During the last decade, the increase in the number of computer applications that rely heavily on spatial data and on multimedia data has caused the database community to focus on the management and retrieval of multidimensional data. Nearest Neighbor (NN) search is important in Geographic Information Systems (GIS) as well as in multimedia applications. For example, in GIS applications, NN search can be used to find the neighbor cities, schools or factories to a given location. In multimedia database fields, NN search can be used to perform similarity search, which is a popular kind of content-based search.

The existing NN search algorithms can be classified into two different groups. One group is $k$-NN search algorithms, where $k$, the number of neighbor objects to be retrieved, is known and fixed in advance. The algorithms in this group recursively traverse the index-tree in depth-first manner and pruning strategies can be used to reduce the number of node accesses.
The other is Incremental NN (INN) search algorithms, which can also be used when the number of neighbor objects to be retrieved is unknown and is not fixed in advance. The INN search algorithms find and report the neighbor objects one by one from the nearest one until the user is satisfied with the search result. The INN search algorithms can also be used for $k$-NN search. That is, it reports the neighbor objects one by one until the $k$-th neighbor object is reported. In fact, the INN search has been regarded as the optimal NN search algorithm because of the minimum number of node accesses [1].

The main contributions of this paper are as follows. (1) For the first time, this paper analyzes performance of the INN search algorithm with $m$ (the number of neighbor objects reported finally), $n$ (the cardinality of database) and $d$ (the dimensionality of the space where the points are located) as parameters. Using our model, dimensionality curse is mathematically revealed for an arbitrary number of NN objects to be retrieved. (2) For the first time, the two key factors of $d_m$ (the distance from the $m$-th NN object to the query point) and $\sigma_h$ (the side length of each node) are estimated using their upper bounds and their lower bounds, which is very helpful to effectiveness of the model, especially in high-dimensional spaces. (3) For the first time, the particularity on the fanout of the root node of R-trees is also taken into account in this study. (4) For the first time, when R-trees are used for point objects the difference between the fanout of the non-leaf nodes and that of the leaf nodes is also taken into account. We note that the objects in feature spaces are generally points. (5) For the first time, our analysis focuses on the number of node accesses (the existing related studies focus only on the number of accessed leaf nodes) and the length of the priority queue (which still not be analyzed yet), which are very important factors on performance of the INN search. The number of accessed leaf nodes is very important for disk-resident indexes because the leaf nodes are often saved in disk. However, for memory-resident indexes, not only the number of accessed leaf nodes but also the number of the other nodes is very important. And now, the memory-resident indexes tend to become more and more, which will be discussed in Section 3. The theoretical analysis is verified by experimental results.

This paper is organized as follows: In Section 2, related studies and the motivation of our investigation are presented. The analytical model for estimating performance of the newest INN search algorithm is presented in Section 3. In Section 4, the model presented in this paper is verified by experiments. The conclusion is drawn in Section 5.

2 Related Works

R-trees are widely used in multi-dimensional databases and they are regarded as being among the best multi-dimensional indexes. As an example, R*-tree (a famous member of R-tree family) is used in this study. In this section, R-tree and INN search on it are explained briefly before giving related work.

2.1. R-trees

An R-tree is a hierarchy of nested $d$-dimensional MBRs (Minimum Bounding Rectangles). MBR is a hyper-rectangle that minimally bounds the objects in the corresponding subtree. Each non-leaf node of R-tree contains an array of entries, each of which consists of a pointer and an MBR. The pointer refers to one child node of this node and the MBR is the minimum bounding rectangle of the child node referred to by the pointer. Each leaf node of R-tree contains an array of entries, each of which consists of an object identifier and its corresponding point (for point-objects) or its MBR (for extended-objects). Each MBR can be indicated by two points. The capacity of each node (except the root node) is usually chosen such that a node fills up one disk page (or a small number of pages). There exist an upper
bound and a lower bound on the number of entries in each non-root node. The upper bound determines the size of each node and the lower bound can guarantee the lowest storage utilization. The number of entries in one node is also called its fanout. The number of entries in root node should be greater than one.

Figure 1 shows an example of R*-tree.

![Figure 1](image)

**Figure 1.** An example of R-tree (a) data space (b) R-tree built from the data in (a).

In an R-tree for point objects (e.g., in feature spaces), the fanout of leaf nodes can be twice as much as that of non-leaf-root nodes. This is because that two points are necessary to indicate each MBR in non-leaf nodes,

### 2.2. INN Search on R-trees

Three incremental solutions to the NN search problem exist in the literature [2, 3, 4]. All of these algorithms employ priority queues. The algorithm [2] employs one priority queue to contain objects and a stack to keep track of the subtrees of the index that have not yet to be completely processed. The algorithm [3] employs two priority queues, one for objects and another for nodes of the index. The algorithm [4] employs only one priority queue for objects and nodes of the index.

The third algorithm [4] is referred to in some other works [5, 6] and is recognized as the newest and fastest one [6]. This INN search has been regarded as the optimal NN search algorithm because of the minimum number of node accesses [1], which is the reason why it is chosen in this study. Throughout the remainder of this paper, the INN search algorithm refers to this one.

The key of the INN search algorithm is the use of one priority queue to contain objects and nodes of the index. The objects and the nodes in the priority queue are sorted in ascending order of their distance values (for objects) or MINDIST (for nodes) from the given query point, where MINDIST is the minimum distance of a node (i.e., its MBR) from the query point [7]. Initially, the priority queue is empty. This algorithm begins with inserting the root node in the priority queue. The members (nodes or objects) of the priority queue are dequeued one by one. If the dequeued member is a non-leaf node, then all of its child nodes are inserted in the priority queue. If the dequeued member is a leaf node, then all of its objects are inserted. This process is a “dequeue-insert” process. The members (objects and nodes) of the priority queue must be sorted in ascending order of their distances from the query point at all times. Note that, when some object(s) and some node(s) have the same distance values from the query object, the object(s) should be located before the nodes. If the dequeued member is an object, this object is reported as the newest neighbor object. The algorithm repeats the “dequeue-insert” process until user is satisfied with the search result or a wanted number of NN objects have been reported. INN algorithm is shown as follows. The efficiency of this INN search algorithm was discussed [6] and its optimality was proven by S. Berchtold et al.[1].
PriorityQueue queue;
queue.insert(0, root);
Wait(Continue or not?);
WHILE not queue.isempty() DO
    first=queue.dequeue();
    CASE first is an intermediate node:
        FOR each child in first DO
            queue.insert(MINDIST(q, child), child);
    CASE first is a leaf node:
        FOR each object O in first DO
            queue.insert(dist(q, O), O);
    CASE first is an object:
        report(first);
        wait(Continue or not?);
ENDWHILE

2.3. Performance Analysis of the INN Search Algorithm

The original idea of the above INN search algorithm is proposed by G.R. Hjaltason and H. Samet [4], who proposed an algorithm for ranking all the objects in one spatial database according to the distances from the objects to a given query point. This algorithm is adapted for INN search by T. Seidl and H.P. Kriegel [5], which reports the neighbor objects one by one from the nearest one until the user is satisfied with the result. Its performance estimation is discussed briefly and tested by G.R. Hjaltason and H. Samet [6]. However, there exist the following four problems in their analysis.

1. The presented model analyzed the following two factors only. (1) The expected number of leaf nodes that intersect with the search region, where the search region means the circle region with the query point as its center and the distance from the last reported neighbor object to the query point as its radius. (2) The expected number of objects remaining in the priority queue. They did not analyze the expected number of node accesses and the length of the priority queue. In fact, accessed nodes do not only include accessed leaf-nodes and the priority queue contains not only objects but also some nodes.

2. The analytical model is for 2-dimensional spaces only. And it can not be simply generalized to high-dimensional spaces.

3. The presented model is not verified by experiments. Many experiments on the INN search performance are performed and discussed with uniformly distributed objects and the real objects as well. However, the presented model is not verified by experiments. That is, the number of leaf nodes that intersect with the search region and the number of objects remaining in the priority queue are mathematically analyzed but they are not tested by experiments.

4. The analysis is based on the following assumption: “on the average, half of the objects in the leaf nodes that intersect with the search region are inside the search region, while half are outside”. We think this assumption is farfetched. Because the situation is expected to vary greatly even for uniformly distributed point objects as the number of neighbor objects reported finally changes.

Stefan Berchtold et al. present a cost model for NN search [1]. However, as pointed out in the conclusion section of that paper, the cost model can be used for 1-NN search only. That is, the cost model can be used only in the case that one NN object is reported and that model can not simply be generalized to the cases that an arbitrary number of NN objects are finally reported. Moreover, it analyzes the number of accessed leaf nodes only and it does not present any analysis on the length of the priority queue, which is an important factor on search performance.
The work [8] only gave a model for estimating the number of node accesses of the NN search algorithm. However, it did not touch on the estimation of the length of the priority queue, which is also very important to the search performance.

Dong-Ho Lee et al. also touch upon this INN algorithm [9]. They propose a new special space partitioning strategy, called SPY-TEC. Somewhat like the pyramid-tree proposed by S. Berchtold and H.P. Kriegel [10], the SPY-TEC partitions the $d$-dimensional space into $2^d$ spherical pyramids having the center point of the space as their top, and the curved $(d-1)$-dimensional surface as their bases. And then they cut each spherical pyramid into several spherical slices, each of which corresponds to a one-dimensional value. Thus, a B$^+$-tree can be used. However, calculating the distances from the query point to a spherical pyramid and to a spherical slice (bounding slice) becomes very complicated. This calculation is necessary for INN search and the calculation of the distance from the query point to a node MBR of R*-tree is much simple. The main contributions of that paper are as follows. (1) It proposed a new special space partitioning strategy; (2) It proposed formulas for calculating the distance from the query point to a spherical pyramid and that from the query point to each spherical slice (or say bounding slice), $MINDIST(q, sp_i)$ and $MINDIST(q, BS_l)$, respectively; (3) The INN search algorithm is slightly adapted to this new space partitioning strategy. However, the main idea of the INN algorithm is not changed at all and its performance analysis is not touched upon.

3. Performance Estimation of the INN Search Algorithm

The number of node accesses is an important factor on search performance. For disk-resident indexes, it is directly related to the number of disk I/O operations; for memory-resident indexes, it is directly related to the number of cache misses.

The priority queue is an important data structure in the INN search algorithm and it is not long in low-dimensional spaces [6]. However, according to our investigations, it is very long and is very expensive to maintain for large databases in high-dimensional spaces, especially for memory-resident indexes. The long priority queue that must keep sorted at all times means a large number of insertions and comparisons.

As the price of memory continues to drop, it is now feasible to place a common index in main memory. During searching on such memory-resident indexes, the traditional bottleneck of disk access becomes not very hard [11, 12].

Based on the above-mentioned reasons, we present the formulas for estimating the number of node accesses and estimating the length of the priority queue, with $m$ (the number of neighbor objects reported finally), $n$ (the cardinality of database) and $d$ (the dimensionality of the space where the points are located) as parameters.

For simplicity, like some other works [6, 12], we assume that both data objects and the query points are uniformly distributed in the domain. Without loss of generality, as in other analytical works [13, 14, 15], we assume that the data domain is a unit hyper-cube and we think that, in this case, it is reasonable to assume that the R-tree nodes of the same height have cube-like MBRs roughly of the same size [6, 12, 13].

Some symbols used in the analysis and their description are shown as follows.

\begin{align*}
Accesses_{node} & : \quad \text{number of node accesses} \\
Lq & : \quad \text{length of the priority queue} \\
d & : \quad \text{dimensionality of database} \\
n & : \quad \text{cardinality of database} \\
m & : \quad \text{number of neighbor objects reported finally} \\
q & : \quad \text{query point} \\
f_l & : \quad \text{average number of entries in each leaf node} \\
f_r & : \quad \text{average number of entries in each non-leaf-root node}
\end{align*}
Note that the levels are counted from the root level whose level number is zero.

Here is the definition of search region.

**Definition (search region):**

We wish to analyze the situation up to \( m \) neighbor objects have been reported. Let \( o \) be the \( m \)-th neighbor object of the query point \( q \), and \( d_m \) is distance from \( o \) to \( q \). The region within distance \( d_m \) from \( q \) is called the search region.

### 3.1. Propositions

Several important propositions are given as follows.

**Proposition 1**

At the moment when the \( m \)-th neighbor object is dequeued, the following equation holds.

\[
N_{h,\text{dequeued}} = N_{h,\text{inter}}, \quad (1)
\]

where \( h \) refers to the level of R-tree. Certainly, \( 0 \leq h \leq H-1 \). \( N_{h,\text{dequeued}} \) refers to the number of nodes in level \( h \) that have already been dequeued from the priority queue. \( N_{h,\text{inter}} \) is the number of nodes in level \( h \) that intersect with or are contained in the search region (i.e., those nodes with \( \text{MINDIST}(q, \text{node}) < d_m \)).

**Proof:**

Remember that all the members of the priority queue are sorted in ascending order of their distances (for objects) or MINDISTs (for nodes) from \( q \) (note that, objects are located before nodes in the priority queue if they have the same distance from \( q \)). And the MINDIST value from \( q \) of any child node of each node, obviously, must be greater than or equal to the MINDIST value of this node. Thus, (1) the MINDIST value of any node dequeued from the queue must be less than \( d_m \). That is, they must intersect with or be contained in the search region, and (2) on the other hand, it is impossible for the nodes whose MINDISTs are less than \( d_m \) still to stay in the queue or to be contained in some node that still stays in the queue. In other words, all the nodes that intersect with or that are contained in the search region must have been dequeued.

Proposition 1 means that the number of nodes in any level that have been dequeued from the queue must be the same as the number of nodes in this level that intersect with the search region.

**Proposition 2**

At the time when the \( m \)-th neighbor object is dequeued, the expected number of nodes in level \( h \) that have been inserted and still remain in the priority queue, \( N_{h,\text{left}} \), is given by
\[ N_{h,\text{left}} = \begin{cases} f_r - N_{h,\text{inter}} & \text{(if } h = 1) \\ f_i \cdot N_{h-1,\text{inter}} - N_{h,\text{inter}} & \text{(if } h > 1) \end{cases} \]  

(2)

Proof:

(1) if \( h > 1 \), at the parent level of level \( h \) (i.e., level \( h-1 \)), there are \( N_{h-1,\text{dequeued}} \) nodes have been dequeued and all of their \( f_i \cdot N_{h-1,\text{dequeued}} \) child nodes (in level \( h \)) have been inserted in the priority queue. Of the nodes in level \( h \) that have been inserted in the queue, \( N_{h,\text{dequeued}} \) nodes have been dequeued. Therefore,

\[ N_{h,\text{left}} = f_i \cdot N_{h-1,\text{dequeued}} - N_{h,\text{dequeued}} = f_i \cdot N_{h-1,\text{inter}} - N_{h,\text{inter}} \]  

(3)

(2) if \( h = 1 \), the parent of this level is the root node and the root node must have already been dequeued from the priority queue. Thus, all the \( f_i \) nodes in this level (the child nodes of the root node) have been inserted. Of these \( f_i \) nodes, \( N_{h,\text{inter}} \) (here \( h = 1 \)) nodes have been dequeued. Of course, in this case, \( N_{h,\text{left}} \) is the difference of \( f_i \) and \( N_{h,\text{inter}} \). ♦

Proposition 3

At the time when the \( m \)-th neighbor object is dequeued, the expected number of objects that still remain in the priority queue, \( N_{\text{objects}} \), is given by

\[ N_{\text{objects}} = f_i \cdot N_{\text{leaf,dequeued}} \cdot m, \]  

(4)

where \( N_{\text{leaf,dequeued}} \) refers to the number of leaf nodes that have been dequeued.

Proof:

Since \( N_{\text{leaf,dequeued}} \) leaf nodes have been dequeued, all the \( f_i \cdot N_{\text{leaf,dequeued}} \) objects in these leaf nodes have been inserted in the queue. Note that \( m \) objects have been dequeued. Thus, the number of remaining objects is \( f_i \cdot N_{\text{leaf,dequeued}} \cdot m \). ♦

Proposition 4

For uniformly distributed query point, the probability of the query point being located in any node is the volume of this node.

Proof is omitted since it is obvious. Note that, according to our assumptions given at the beginning of this section, the volume of the whole space is 1.

3.2 Expected Distance from the Query Point to the \( m \)-th NN Object (i.e., \( d_m \))

Since \( d_m \) and \( \sigma_0 \) are two key factors in our model, they are estimated in this and the next subsection, respectively.

3.2.1 Coarse Estimation: See Fig.2. The shadow region, called \( \text{RatioSphere} \), is the hypersphere whose center is \( q \) (the query point) and whose volume is \( m/n \). Its radius is denoted as \( \delta_m \). Considering the volume of the whole space is 1 and all the objects are uniformly distributed, the expected number of objects in the shadow region should be \( m \), which means that \( \delta_m \) can be used to roughly estimate \( d_m \).
Figure 2: Estimation of $d_m$.

Let $Vol_{region}$ be the volume of $RatioSphere$. According to the knowledge of geometry, the volume of $RatioSphere$, $Vol_{region}$, is given by

$$Vol_{region} = \frac{\sqrt{\pi} \delta_m^d}{\Gamma(d/2 + 1)}.$$  \hspace{1cm} (5)

where $\Gamma(x+1) = x \cdot \Gamma(x)$, $\Gamma(1) = 1$, $\Gamma(1/2) = \sqrt{\pi}$.

Considering $Vol_{region} = m/n$, $\delta_m$ can be estimated as follows:

$$\delta_m = d \frac{m \Gamma(d/2 + 1)}{n \pi^{d/2}}.$$ \hspace{1cm} (6)

3.2.2 Revised Estimation: When the Equation (6) is used to estimate $d_m$, the question may occur “Is the $m$-th NN object of $q$ really just located on the surface of $RatioSphere$?” If not so, then $d_m$ may be less than $\delta_m$. See Fig.2 (b).

However, $d_m$ is not less than $\delta_{m-1}$. This is because that, if $d_m < \delta_{m-1}$ then the expected number of objects in the search region is less than or equal to $m-1$. That is,

$$\delta_{m-1} \leq d_m \leq \delta_m.$$ \hspace{1cm} (7)

Considering that the objects are uniformly distributed, the average value of $\delta_m$ and $\delta_{m-1}$ is used as $d_m$. That is,

$$d_m = \frac{1}{2} (\delta_m + \delta_{m-1}) = \frac{d \sqrt{m + (d/2-1)}}{n \sqrt{\pi}}.$$ \hspace{1cm} (8)

We noticed that the difference between $\delta_{m-1}$ and $\delta_m$ and the difference between $\delta_m$ and $d_m$ should not be ignored, especially in high-dimensional spaces. This is because the objects are very sparse in high-dimensional spaces. The above differences are still not considered in the existing analyzing works which suffer from dimensionality curse greatly. Because $d_m$ is a key factor to our performance model, we believe that the revising of $d_m$ estimation is very helpful to our model, especially in high-dimensional spaces.
3.3 Expected length of the Side of Node MBR

It is clear that the expected number of nodes in level $h$, $n_h$, can be given by

$$n_h = f_r \cdot f_l^{h-1} \quad (h \geq 1),$$

(9)

The expected number of objects in each node of level $h$ is $n/n_h$. Let us consider a subspace (called ShareSpace) with a volume of $1/n_h$. Considering the objects are uniformly distributed and the volume of the whole space is 1, the expected number of objects in ShareSpace is $n/n_h$, which is just the same as the expected number of objects in each node of level $h$. Therefore, ShareSpace can be used to estimate the size of each node in level $h$. The size of ShareSpace is the upper bound on the size of each node in level $h$ since there are not always objects on all the faces of ShareSpace. That is,

$$Vol_{node} = \sigma_h^d \leq \frac{n}{n_h} = \frac{1}{n_h},$$

(10)

If the number of objects in ShareSpace is reduced by 1, then the size of the new ShareSpace can give the lower bound on the size of each node in level $h$. That is,

$$Vol_{node} = \sigma_h^d \geq \frac{n - 1}{n_h} = \frac{1}{n_h} - \frac{1}{n}.$$

(11)

Considering the uniformly distribution, $\sigma_h$ in this study uses the average of its upper bound and its lower bound. That is,

$$\sigma_h = \frac{1}{2} \left( \frac{1}{n} \right)^d \left[ \frac{n}{f_r \cdot f_l^{h-1}} \cdot \frac{1}{d} \left( f_l - 1 \right) + \frac{n}{f_r \cdot f_l^{h-1}} \cdot \frac{1}{d} \right].$$

(12)

Since this number of each leaf node is $f_l$, the side length of each leaf node, $\sigma_l$, can be given by

$$\sigma_l = \frac{1}{2} \left( \frac{1}{n} \right)^d \left[ \left( f_l - 1 \right) \cdot \frac{1}{d} + f_l^d \cdot \frac{1}{d} \right].$$

(13)

In this way, $\sigma_h$ and $\sigma_l$ are estimated using the average of their upper bounds and their lower bounds instead of using the share space only. Because $\sigma_h$ and $\sigma_l$ are also key factors to our performance model, we believe that such an estimation of them is helpful to our model, especially in high-dimensional spaces where the objects are very sparse.
3.4 Expected Number of Node Accesses

**Proposition 5**

The probability of the search region intersecting with or being contained in any node of level \( h \), \( P_{h,\text{intersect}} \), is given by

\[
\tau = \sum_{i=0}^{d} \binom{d}{i} \cdot \sigma_h^{d-i} \cdot \frac{\sqrt{\pi^i}}{\Gamma(i/2 + 1)} \cdot d_m^i,
\]

\[
P_{h,\text{intersect}} = \min\{\tau, 1\},
\]

where \( d_m \) is estimated by Equation (8) and \( \sigma_h \) can be given by Equation (12).

**Proof:**

See Fig. 3. The rectangle is a node MBR in level \( h \) whose side length is \( \sigma_h \). The dotted circle has the same size as the search region and touches the side of the node MBR. The round-corner rectangle (called Minkowski-sum) is the trace of the center of the dotted circle after the dotted circle makes a circuit, keeping the touching state, along the sides of the node MBR.

![Figure 3. Example of Minkowski-sum in 2-dimensional space.](image)

It is clear that, if the search region intersects with the MBR of this node, then the center of the search region, \( q \), is located in the Minkowski-sum of this node and vice versa. That is,

\[
P_{h,\text{intersect}} = P_{q,\text{mink}} = \text{Vol}_{\text{mink}},
\]

where \( P_{q,\text{mink}} \) refers to the probability that \( q \) is contained in the Minkowski-sum; \( \text{Vol}_{\text{mink}} \) is the volume of Minkowski-sum. Thus, calculating the volume of Minkowski-sum in \( d \)-dimensional space is necessary for calculating \( P_{h,\text{intersect}} \).

The following general method to calculate the volume of Minkowski-sum in \( d \)-dimensional space has been proposed and mathematically proved by ChangZhou Wang and X. Sean Wang [16].

\[
\text{Vol}_{\text{mink}} = \sum_{i=0}^{d} Q_i \cdot \sum_{\{k_i : k_i \leq L\}} \prod_{j \in \{k_i, L\}} a_j,
\]

where

\[
\sum_{i=0}^{d} Q_i = m^n.
\]
where $Q_i$ refers to the volume of the search region in $i$-dimensional space; \[
\sum_{1 \leq k \leq l} \frac{i}{k}
\] means “for all possible ways of selecting $i$ dimensions from a total number of $d$ dimensions without regard to the order”. This is clearly a problem of combination and there are totally \( \binom{d}{i} \) possible ways of selecting. $a_j$ refers to the side length of the node rectangle in the $j$-th dimension.

According to the assumptions in this study, $a_j = \sigma_h$ for all $1 \leq j \leq d$. Thus, in this study,

\[
Vol_{\text{min}} = \sum_{i=0}^{d} Q_i \left( \binom{d}{i} \right) \sigma_h^{d-i}, \quad Q_i = \frac{\sqrt{\frac{\pi^i}{\Gamma(i/2+1)}}}{\Gamma(i/2+1)} \cdot d_i^m.
\]  

(17)

If $P_{\text{intersect}} \cdot Vol_{\text{min}} \leq 1$ is considered, Proposition 5 can be proved.

Lemma 1: the expected number of nodes in level $h$ that intersect with or are contained in the search region, $N_{h, \text{inter}}$, can be given by

\[
N_{h, \text{inter}} = n_h \cdot P_{h, \text{intersect}}.
\]

(18)

where $P_{h, \text{intersect}}$ is estimated by Proposition 5 and $n_h$ can be given by Equation (9).

Proof: To calculate $N_{h, \text{inter}}$ we have to sum $P_{h, \text{intersect}}$ for every node in this level. Since the nodes in this level have the same probability of intersecting with or being contained in the search region, $N_{h, \text{inter}}$ can be given by multiplying $P_{h, \text{intersect}}$ with the number of nodes in this level, $n_h$.

According to Proposition 1 and considering that the root node must be accessed, by the moment when the $m$-th neighbor object is reported, the expected number of node accesses (i.e., the number of nodes that have been dequeued from the queue), $\text{Accesses}_{\text{node}}$, is given by

\[
\text{Accesses}_{\text{node}} = 1 + \sum_{h=1}^{H} N_{h, \text{inter}}.
\]

(19)

If the corresponding equations are substituted into the above Equation (19), the estimating formula of the node accesses can be given by Equation (20).

\[
\text{Accesses}_{\text{node}} = 1 + \sum_{h=1}^{H} f_r \cdot j_h^{-1} \cdot \min \left\{ \frac{d}{i}, \sum_{i=0}^{d} \left( \binom{d}{i} \cdot \frac{\pi^i \cdot d_i^m \cdot \sigma_h^{d-i}}{\Gamma(i/2+1)} \right) \right\},
\]

(20)

where $d_m$ and $\sigma_h$ are given by Equations (8) and (12), respectively.

3.5 Expected Length of the Priority Queue

Using the priority queue is a distinctive feature of the INN algorithm and the priority queue
is not long in low-dimensional spaces. However, according to our investigations, the priority queue may be very long for large databases in high-dimensional spaces. It is clear that maintaining a very-long sorted queue means a large number of comparisons and insertions. Thus, the length of the priority queue is an important factor on performance of the INN search algorithm. However, analyzing on the length of the priority queue has not been done yet. Our model for estimating the length of the priority queue is presented in this subsection.

Generally, when the \( m \)-th neighbor object is dequeued, there exist both some nodes and some objects remaining in the queue unless \( m \) is very large. Thus, after the \( m \)-th neighbor object is dequeued,

\[
L_q = N_{\text{object}} + N_{\text{node}},
\]

(21)

where \( N_{\text{object}} \) and \( N_{\text{node}} \) refer to the number of objects and the number of nodes that still remain in the queue when the \( m \)-th neighbor object is dequeued, respectively.

According to Proposition 5, the probability of the search region intersecting with or being contained in any leaf node, \( P_{l,\text{intersect}} \), can be given by

\[
P_{l,\text{intersect}} = \min\{\tau_l, 1\},
\]

(22)

where \( \tau_l \) can be given by Equation (13).

According to Proposition 1 and Equation (18), the number of leaf nodes that have been dequeued, \( N_{\text{leaf,dequeued}} \), can be given by

\[
N_{\text{leaf,dequeued}} = N_{\text{leaf,inter}} = \text{Num}_{\text{leaf}} \cdot P_{l,\text{intersect}} = n/f \cdot P_{l,\text{intersect}},
\]

(23)

where \( N_{\text{leaf,inter}} \) refers to the number of leaf nodes that intersect with the search region. \( P_{l,\text{intersect}} \) can be given by Equation (22).

Firstly, let us see \( N_{\text{object}} \) in Equation (21). According to Proposition 3 and Equation (23),

\[
N_{\text{object}} = f \cdot N_{\text{leaf,dequeued}} \cdot m = n \cdot P_{l,\text{intersect}} - m.
\]

(24)

We can understand Equation (24) as follows: when the \( m \)-th neighbor object is dequeued from the queue, \( N_{\text{leaf,dequeued}} \) leaf nodes have already been dequeued and all of their objects (the total number is \( f \cdot N_{\text{leaf,dequeued}} \)) are inserted in the queue. \( m \) means the number of objects that have already dequeued from the queue.

Then, let us estimate \( N_{\text{node}} \) in Equation (21). Proposition 2 and Equation (18) present the calculating method for \( N_{\text{h,left}} \). If we sum \( N_{\text{h,left}} \) for each level except root level, the expected number of nodes left in the priority queue when the \( m \)-th neighbor object is dequeued, \( N_{\text{node}} \), can be given by

\[
N_{\text{node}} = \sum_{h=1}^{H-1} N_{\text{h,left}},
\]

(25)
where $N_{h,\text{left}}$ can be given by Proposition 2.

By substituting Equations (25), (24) and the other corresponding equations in Equation (21), the length (the number of members) of the priority queue when the $m$-th neighbor object is obtained can be estimated by Equation (26). Note that $f_r$, $f_i$, $f_l$ and $H$ in Equation (26) will be discussed later.

Note that the above formulas are based on the average case and not absolute ones. However, we are interested in the average case, and exceptional cases do not harm the generality.

3.6 Discussion of $f_r$, $f_i$, $f_l$ and $H$

$f_r$, $f_i$, $f_l$ and $H$ in the above estimating equations are still not discussed. Let $f_r$, $f_l$ denote the maximum number of entries in each non-leaf node and the maximum number of entries in each leaf node, respectively. When one R*-tree is built, there are two possibilities that $F_l=2\times F_i$ (for extended objects or point objects) and $F_l=2\times F_i$, (for point objects). Here the two cases are discussed separately.

3.6.1 Case of $F_l=2\times F_i$: According to the analysis made by C. Faloutsos and I. Kamel [15],

$$f_l = F_i \cdot u, \quad f_l = F_i \cdot u,$$

where $u$ is the average node utilization (typically, 70% for the R*-tree [15]) and, $F_i$ and $F_l$ are given by users. That is,

$$f_i = 0.7 \times F_i; \quad f_l = 0.7 \times F_l \quad F_i = 2 \times F_l$$  

(28)
Thus,
\[ f_l = 2f_i, \]
\[ 1 < f_r \leq f_i \]  
(29)

Clearly,
\[ n = f_r \cdot f_i^{(H-2)} \cdot f_i = 2f_r \cdot f_i^{(H-1)} \]  
(30)

If Equation (29) is considered, then
\[ 2f_i^{(H-1)} \leq n \leq 2f_i^H \]  
(31)

\[ \log f_i (n/2) \leq H < \log f_i (n/2) + 1 \]  
(32)

That means
\[ H = \log_{f_i} (n/2) = \log_{(0.7f_i)} (n/2) \]  
(33)

Considering Equation (30),
\[ f_r = \frac{n}{2(0.7f_i)^{H-1}} \]  
(34)

3.6.2 Case of \( F_l=F_i \) (say \( F \)): In the same way as above, in this case,
\[ f_l = f_i = 0.7F, \quad H = \log f_i (n), \quad f_r = \frac{n}{f_i^{H-1}}. \]  
(35)

4. Experimental Evaluation

Using uniformly distributed points we verified the estimating model presented in this paper as the three parameters (i.e., \( d, n \) and \( m \)) change.

Since the analysis in this paper is based on the average case, the results are the average values of 100 random trials. Note that only the results of the case that \( F_l=2F_i \) is presented in this paper. \( F_i \) is denoted as Fanout in this section. Anyway, according to our study in the other case (\( F_l=F_i \)), the tendency of performance and the error rate of our model have not much change.

4.1 Evaluation with Different Dimensionalities

Without loss of generality, we let \( m = 40 \) and \( n = 40000 \). Since it is well known that R*-tree is not suitable for very-high-dimensional spaces, which is regarded as dimensionality curse, we performed our experiments with \( d \) from 2 to 10. Table 1 shows the calculated results and their experimental counterparts. Note that Fanout is the maximum fanout of the non-leaf nodes.

From Table 1, the following observations can be obtained.

1. Performance of the INN algorithm degrades exponentially as \( d \) increases. In other words, the number of node accesses and the length of the priority queue increase as dimensionality increases. This is consistent with the observation called dimensionality curse mentioned in...
many works on the NN search. Thus, we can say that, using our model, dimensionality curse of NN search is mathematically revealed. Dimensionality curse indicates that NN search performs worse in higher dimensional spaces. We can understand this as follows.

Table 1. Evaluation as d increases (n=40000, m=40).

<table>
<thead>
<tr>
<th>d</th>
<th>Fanout</th>
<th>Access_node</th>
<th>Lq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calculated</td>
<td>tested</td>
<td>error rate (%)</td>
</tr>
<tr>
<td>2</td>
<td>40</td>
<td>5.97</td>
<td>5.81</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>36.80</td>
<td>35.96</td>
</tr>
<tr>
<td>6</td>
<td>13</td>
<td>239.56</td>
<td>232.32</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>1385.84</td>
<td>1302.41</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>4348.04</td>
<td>3671.74</td>
</tr>
</tbody>
</table>

As the dimensionality grows, the distribution of objects becomes more and more sparse. The objects and the node MBRs will become more and more distant from the query object. However, the node MBRs become more distant from the query object much slower than the objects. This means that the search region expands much quickly than the node MBRs become distant from the query point as the dimensionality grows. Thus, the number of nodes that intersect with or contained in the search region will become larger as the dimensionality grows. This means that the number of nodes that have to be accessed will increases and the number of nodes and objects that are inserted in the priority queue also increases in this case.

2. As dimensionality increases, the gap between calculated result and tested result gets large. We think this is because in high-dimensional spaces, the objects become very sparse and it seems that some other factor(s) should be taken into account in very high dimensional spaces.

3. If the dimensionality reaches 10, both the calculated result and the tested result of $L_q$ are close to the total number of the database objects. That is, almost all the objects of databases and almost all the nodes of R-trees have to be accessed in this case, which also mathematically reveal the well-known fact that performance of the INN search algorithm on R-trees becomes worse than sequential scan in high-dimensional spaces.

The issue occurs that how Fanouts are determined in our experiments and its influence on the error rate of our model. In fact, according to our investigations, Fanout only decides the size of nodes and it has no influence on the accuracy of our model. Anyway, we choose Fanouts such that the product of Fanout and d keeps roughly fixed in the experiments presented in this paper, aiming at keeping the node size fixed in different-dimensional spaces. In this way, we can observe the performance tendency of the INN search as the dimensionality changes.

4.2 Evaluation with Different m

We also tested the behavior of our model as the number of m increases. The tested results are shown in Table 2 along with the calculated results. The experiments were performed with $d=4$ and $n=40,000$.

Table 2. Evaluation as m grows (d=4, n=40,000, Fanout=20).

<table>
<thead>
<tr>
<th>m</th>
<th>Access_node</th>
<th>Lq</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calculated</td>
<td>tested</td>
</tr>
<tr>
<td>20</td>
<td>27.27</td>
<td>26.21</td>
</tr>
<tr>
<td>40</td>
<td>36.80</td>
<td>35.96</td>
</tr>
<tr>
<td>60</td>
<td>44.45</td>
<td>43.01</td>
</tr>
<tr>
<td>80</td>
<td>51.14</td>
<td>49.96</td>
</tr>
<tr>
<td>100</td>
<td>57.21</td>
<td>55.34</td>
</tr>
</tbody>
</table>
From Table 2, we know that the change of $m$ has not much influence on the degree of accuracy of our model when $m$ is relatively very small to the cardinality of the database. Another observation is that performance of the INN search algorithm degrades as $m$ increases. We think this is easy to understand.

4.3 Evaluation with Different Cardinalities of Databases

In order to test our model as the cardinality of database increases, we changed our database cardinality from 200 to 200,000. The tested results are shown in Table 3 along with the calculated results. The experiments were performed with $d=4$, $m=40$ and $Fanout=20$.

From Table 3, we observe that

1. The gap between the calculated result and the tested result tends to become smaller as the database becomes larger. We think this is because that larger databases of uniformly distributed points tend to meet well the assumptions in our analysis.

2. If $n$ reaches some number (20000 in Table 4), the number of node accesses and the length of the priority queue increase very slowly as the cardinality of database increases. We think this is because there are the following two contrary factors in this case that counteract each other to some extent.

   (1) On the one hand, as $n$ grows, the point density increases and the node MBRs become smaller. Thus, the number of nodes and the number of leaf nodes that are contained in or intersect with the search region tend to increase.

   (2) On the other hand, as the point density increases, the $d_m$ tends to become shorter. So the volume of the search region becomes smaller, which make the number of nodes and the number of leaf nodes that are contained in or intersect with the search region tend to decrease.

From all above results, we observe that the tested results are generally close to the calculated results, which means that the actual performance of the INN search for uniformly distributed objects is mathematically verified. In other words, one could use the model presented in this paper to estimate performance of INN search for uniformly distributed objects.

Table 3. Evaluation as cardinality increases ($d=4$, $m=40$, $Fanout=20$).

<table>
<thead>
<tr>
<th>$n$</th>
<th>$Access_{node}$</th>
<th></th>
<th>$L_q$</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>calculated</td>
<td>tested</td>
<td>error rate (%)</td>
<td>calculated</td>
<td>tested</td>
</tr>
<tr>
<td>200</td>
<td>8.14</td>
<td>7.26</td>
<td>10.84</td>
<td>159.88</td>
<td>142.01</td>
</tr>
<tr>
<td>2000</td>
<td>30.77</td>
<td>29.02</td>
<td>6.84</td>
<td>697.41</td>
<td>648.32</td>
</tr>
<tr>
<td>20000</td>
<td>36.60</td>
<td>35.51</td>
<td>4.37</td>
<td>774.35</td>
<td>744.98</td>
</tr>
<tr>
<td>40000</td>
<td>36.80</td>
<td>35.96</td>
<td>3.42</td>
<td>777.99</td>
<td>751.27</td>
</tr>
<tr>
<td>60000</td>
<td>36.81</td>
<td>36.13</td>
<td>3.00</td>
<td>781.64</td>
<td>758.10</td>
</tr>
<tr>
<td>80000</td>
<td>37.84</td>
<td>37.21</td>
<td>2.79</td>
<td>785.28</td>
<td>768.21</td>
</tr>
<tr>
<td>100000</td>
<td>38.10</td>
<td>37.75</td>
<td>2.04</td>
<td>788.93</td>
<td>773.19</td>
</tr>
<tr>
<td>200000</td>
<td>38.82</td>
<td>38.54</td>
<td>1.83</td>
<td>799.55</td>
<td>786.04</td>
</tr>
</tbody>
</table>

5. Conclusion

In this paper we proposed a model for uniformly distributed point data to mathematically analyze performance of the INN search algorithm with $m$ (the number of neighbor objects reported finally), $n$ (the cardinality of database) and $d$ (the dimensionality) as parameters, focusing on the number of accessed nodes (not only the number of the accesses leaf nodes) and the length of the priority queue. Using our model, dimensionality curse is mathematically
revealed for an arbitrary number of NN objects retrieved. In this model, the two key factors of the distance from the \textit{m}-th NN object to the query point and the side length of each node are estimated using their upper bounds and their lower bounds. And, the difference of fanouts among the leaf nodes, the root node and the other nodes is taken into account.

6. Reference

Authors

YaoKai FENG  received his B.E. degree and M.E. degree in Computer Eng. and Science from Tianjin University, China, in 1986 and 1992, respectively. And, since he received Ph.D. degree in Information science from Kyushu University, Japan, in 2003, he has been with the same university as an assistant professor. His research interests include multidimensional indexing, spatial databases, time series data, and XML databases. He is a member of IPSJ, IEEE and ACM.

Kunihiko KANEKO  received his Ph.D degree from Kyushu University, Japan. Since 1996, he has been with the Graduate School of Information Science and Electrical Engineering, Kyushu University, where he is an associate professor. His research interests include spatial databases, and biomedical databases. He is a member of IPSJ, IEICE, ACM and IEEE.

Akifumi MAKINOUCHI  received his B.E. degree from Kyoto University, Japan, in 1967, Docteur-ingereur degree from Univercite de Grenoble, France, in 1979, and D.E. degree from Kyoto University, Japan, 1988. From 1990 to 2006, he was with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan, where he is a professor. Since 2006, he has been with the Department of Information Network Engineering, Kurume Institute of Technology, Japan. He is a member of IEICE, IPSJ, ACM, and IEEE.