An Estimating Model of Node Accesses for INN Search Algorithm in Multidimensional Spaces

Yaokai Feng [*] Akifumi Makinouchi [†] Kunihiko Kaneko [‡]

Abstract— **Nearest Neighbor (NN) search has been widely used in spatial databases and multimedia databases. Incremental NN (INN) search 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. R*-tree is still regarded as being among the best high-dimensional indices. This paper presents an analytical model for estimating performance of the INN search algorithm on R*-tree. For the first time, our model takes m (the number of neighbor objects reported finally), n (database cardinality) and d (dimensionality) as parameters, focusing on the number of node accesses. In our model, (1) the two key factors of d_m (distance from the m-th NN object to the query point) and σ_h (side length of each node) are estimated using their upper bounds and their lower bounds, which is very helpful to effectiveness of our model; (2) the particularity on the number of entries in the root node and the possible difference of fanouts between the leaf nodes and the other nodes are taken into account. The theoretical analysis is verified by experiments.**

Keywords: INN Algorithm, multidimensional index, R*-tree, node accesses, multidimensional spaces

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 very 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 very 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 strategy can be used to reduce the number of node accesses. The other is Incremental NN (INN) search algorithms, which can be used when the number of neighbor objects to be retrieved is unknown in advance. The INN search algorithm in [1] has been regarded as the optimal NN search algorithm [2].

R*-tree[3] is selected in this paper because it is still regarded as being among the best multi-dimensional indexes and is widely used in researches and application systems related to multi-dimensional databases.

The main contributions of this paper are as follows. (1) This paper analyzes performance of the INN search algorithm with m (the number of neighbor objects reported finally), n (database cardinality) and d (dimensionality) as parameters. (2) The two key factors of d_m (distance from the m-th NN object to the query point) and σ_h (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) The particularity on the number of entries in the R*-tree root-node is taken into account. (4) 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. (5) Our analysis focuses on the number of node accesses (only the number of accessed leaf nodes is analyzed in the other relevant works), which is a very important factor on performance of the INN search. Note that, although the number of accessed leaf nodes is a very important factor of search performance, only the number of leaf node accesses is not enough for many cases.

This paper is organized as follows. In Section 2, related studies and the motivation 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 experi-

[*]Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka City, Japan. Email: fengyk@is.kyushu-u.ac.jp

[†]Department of Information Network Engineering, Kurume Institute of Technology, Japan. Email: akifumi@cc.kurume-it.ac.jp

[‡]Graduate School of Information Science and Electrical Engineering, Kyushu University, Fukuoka City, Japan. Email: kaneko@is.kyushu-u.ac.jp

Manuscript received May 12, 2007.

ments. The conclusion is drawn in Section 5.

2 Related Work

2.1 INN Search on R-trees

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 $MINDIST$s (for nodes) from the given query point, where $MINDIST$ is the minimum distance of a node (i.e., its MBR: Minimum Bounding Rectangle) from the query point [4]. 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 untill user is satisfied with the search result or a wanted number of NN objects have been reported. The INN search algorithm is shown in Fig.1.

```
Algorithm INN::INN_search(QueryObject:q)
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
```

Figure 1: **INN algorithm.**

2.2 Performance Analysis of the INN Search Algorithm

Performance estimation of the INN search algorithm is discussed briefly and tested in [1]. However, there exist the following four problems in their analysis.

1. The presented model is on the following two factors only. (1) The expected number of leaf nodes that intersect the search region (defined later) and (2) the expected number of objects remaining in the priority queue. However, they did not analyze the expected number of node accesses. Certainly, accessed nodes do not only include leaf-nodes and the priority queue contains not only objects.

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

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 (defined later) 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 [2]. However, as pointed out in the conclusion section of that paper, that cost model can be used only in the case that one NN object is reported and that model can not simply be generalized to an arbitrary number of reported NN objects. We noticed that only one NN object is not enough in most applications. Moreover, it analyzes the number of accessed leaf nodes only.

There are still some performance analytic works for some other search algorithms, including the work [5] for k-NN algorithm when k=1 and the work [6] for range search algorithm.

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.

For simplicity, like some other works [1, 8], 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 [5, 6, 9], 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 [1, 5, 8].

Some symbols used in the analysis and their descriptions are shown in Table 1.

Table 1: some symbols and their descriptions.

Symbol	Description
$Accesses_{node}$	number of node accesses
f_i	average number of entries in each non-leaf-root node
f_r	number of entries in root node
d	dimensionality of database
d_m	distance from the m-th neighbor object to the query point
n	cardinality of database
σ_h	side of the MBR of each node in level h
m	number of neighbor objects reported finally
σ_l	side of the MBR of each leaf node
q	query point
n_h	number of nodes in level h
f_l	average number of entries in each leaf node
H	Height of index tree

Note that the level numbers are counted from the root level whose level number is zero.

In this paper, we present a model for estimating the number of node accesses of the INN search algorithm with m, n and d as parameters.

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.* See Fig.2.

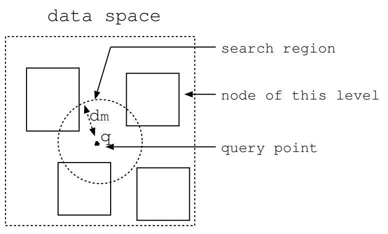

Figure 2: Nodes in level h that intersect with the search region.

Then, let us consider the appearance of the priority queue when the m-th neighbor object is dequeued (or say, obtained), which is shown in Fig.3. In this figure, the black dots in the priority queue are objects and the rectangles are nodes. Generally speaking, when the m-th neighbor object is dequeued, there exist some nodes and objects remaining in the queue unless m is very large. Several important propositions are given as follows.

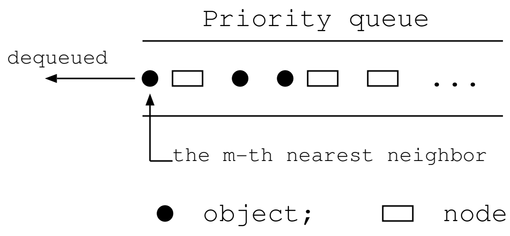

Figure 3: Priority queue when the m-th nearest neighbor is dequeued.

Proposition 1

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

$$N_{h,dequeued} = N_{h,inter}$$

where h refers to the level of R-tree and $0 \leq h \leq H-1$. $N_{h,dequeued}$ refers to the number of the nodes in level h that have already been dequeued from the priority queue. $N_{h,inter}$ is the number of the nodes in level h that intersect with the search region (i.e., those nodes with $MINDIST(q, node) < d_m$). In Fig.2, the region of the dotted circle is the search region and the rectangles are nodes in level h.

Proof:

Remember that all the members of the priority queue are sorted in ascending order of their distances (for objects) or $MINDIST$s (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 that has been dequeued from the queue must be less than d_m. That is, they must intersect with the search region, and

(2) on the other hand, it is impossible for the nodes whose $MINDIST$s 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 the search region must have been dequeued.

Thus, Proposition 1 is true. □

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 the nodes in level h that have been inserted and still remain in the priority queue, $N_{h,left}$, is given by

$$N_{h,left} = \begin{cases} f_r - N_{h,inter} & (if \;\; h = 1) \\ f_i \cdot N_{h-1,inter} - N_{h,inter} & (if \;\; h > 1) \end{cases}$$

Proof:

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

$$\begin{aligned} N_{h,left} &= f_i \cdot N_{h-1,dequeued} - N_{h,dequeued} \\ &= f_i \cdot N_{h-1,inter} - N_{h,inter} \end{aligned}$$

(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_r nodes in this level (child nodes of the root) have been inserted. Of these f_r nodes, $N_{h,inter}$ (here $h = 1$) nodes have been dequeued. Thus, $N_{h,left}$ is the difference of f_r and $N_{h,inter}$ □

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_{object}, is given by

$$N_{object} = f_l \cdot N_{leaf,dequeued} - m$$

where $N_{leaf,dequeued}$ refers to the number of the leaf nodes that have been dequeued.

Proof:

Since $N_{leaf,dequeued}$ leaf nodes have been dequeued, all the $f_l \cdot N_{leaf,dequeued}$ 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 $f_l \cdot N_{leaf,dequeued} - m$. □

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:

See Fig.4. It is clear that, if the possibility of the query point being located in any location keeps the same, the probability of the query point being contained in one node, P_{node}, is given by

$$P_{node} = \frac{Volume_{node}}{Volume_{space}}$$

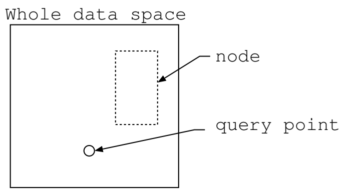

Figure 4: Probability of the query point being located in any node.

Considering the volume of the whole space is 1. Thus, Proposition 4 is true. □

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

3.1.1 Coarse Estimation

See Fig.5(a). The shadow hyper-sphere is called *RatioSphere* whose center is q and volume is $\frac{m}{n}$. Its radius is denoted as δ_m. Considering the volume of the whole space is 1 and all the objects are uniformly distributed, the expected number of objects in *RatioSphere* should be m, which means that δ_m can be used to roughly estimate d_m. Obviously, *RatioSphere* is the average space share of m NN objects in the whole space.

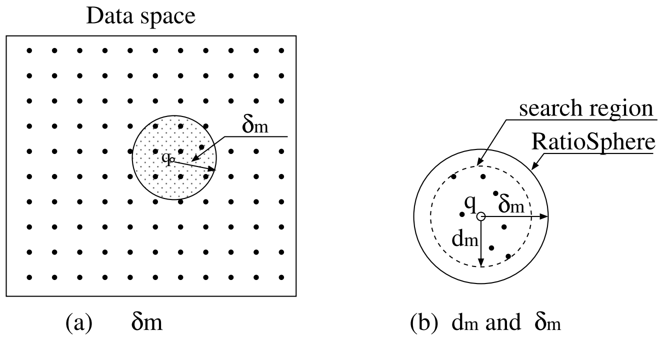

Figure 5: Estimation of d_m.

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

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

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

Considering $Vol_{region} = \frac{m}{n}$, δ_m can be estimated by

$$\delta_m = \sqrt[d]{\frac{m}{n} \cdot \frac{\Gamma(d/2+1)}{\sqrt{\pi}^d}} \tag{1}$$

3.1.2 Revised Estimation

When Equation (1) is used to estimate d_m, the question may occur that "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 δ_m. See Fig.5(b).

However, d_m is not less than δ_{m-1}. This is because that, if $d_m < \delta_{m-1}$ then the expected number of objects in the search region (defined in Page 3) is less than or equal to $m-1$, which is less than m. That is,

$$\delta_{m-1} \leq d_m \leq \delta_m \tag{2}$$

Considering that the objects are uniformly distributed, the average of δ_m and δ_{m-1} is used as d_m. That is,

$$d_m = \frac{1}{2}(\delta_m + \delta_{m-1}) = \frac{1}{2\sqrt{\pi}} \sqrt[d]{\frac{\Gamma(d/2+1)}{n}} \cdot (\sqrt[d]{m} + \sqrt[d]{m-1}) \tag{3}$$

We noticed that the difference between δ_{m-1} and δ_m and, the difference between δ_m and d_m should not be ignored, especially in high-dimensional spaces where the objects are very sparse. The above differences are still not considered in the existing analyzing works. Because d_m is a key factor to the performance model, we believe that the revising of d_m estimation is very helpful to our model.

3.2 Expected the length of the Side of Node MBR (i.e., σ_h)

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

$$n_h = f_r \cdot f_i^{h-1} (h \geq 1) \tag{4}$$

Let *ShareSpace*(k) refer to the average space share of k NN objects in the whole space, whose volume should be k/n since the objects are uniformly distributed and the volume of the whole space is 1. The expected number of objects in each node of level h is n/n_h. Thus, the corresponding space share of each node in level h is $ShareSpace(n/n_h)$, whose volume, Vol_{share}, is

$$Vol_{share} = \frac{\frac{n}{n_h}}{n} = \frac{1}{n_h}$$

Clearly, Vol_{share} can be used to estimate the expected volume of each node. However, it is only the upper bound on the volume of each node in level h since there is not always objects on all the faces of the corresponding $ShareSpace$ of each node.

On the other hand, let us consider the corresponding space share if the number of objects in each node is reduced by 1. That is $ShareSpace(n/n_h - 1)$, whose volume, Vol'_{share}, is

$$Vol'_{share} = \frac{\frac{n}{n_h} - 1}{n}$$

Like in Section 3.1, Vol'_{share} can give the lower bound on the volume of each node in level h. This is because that if the volume of each node is less than Vol'_{share} then the expected number of objects in this node should be less than or equal $n/n_h - 1$, which is less than the expected number of objects in each node.

The expected volume of each node in level h, Vol_{node} can be given by

$$Vol_{node} = \sigma_h^d = \frac{1}{2}(Vol_{share} + Vol'_{share})$$

If Equation (4) is substituted, then σ_h can be given by

$$\sigma_h = \frac{1}{2}\left(\frac{1}{n}\right)^{\frac{1}{d}} \left[\left(\frac{n}{f_r \cdot f_i^{h-1}} - 1\right)^{\frac{1}{d}} + \left(\frac{n}{f_r \cdot f_i^{h-1}}\right)^{\frac{1}{d}}\right] \tag{5}$$

Considering the expected number of the objects in each leaf node is f_l, σ_l can be given by

$$\sigma_l = \frac{1}{2}\left(\frac{1}{n}\right)^{\frac{1}{d}} \left[(f_l - 1)^{\frac{1}{d}} + f_l^{\frac{1}{d}}\right] \tag{6}$$

In this way, σ_h and σ_l are estimated using the average of their upper bounds and their lower bounds instead of using the share space of each node only. Because σ_h and σ_l are also key factors to the performance model, we believe that such an estimation of them is helpful to our model.

3.3 Expected Number of Node Accesses

Proposition 5

The probability of the search region intersecting with any node of level h, $P_{h,intersect}$, is given by

$$\tau = \sum_{i=0}^{d} \left(\begin{array}{c} d \\ i \end{array} \right) \cdot \sigma_h^{(d-i)} \cdot \frac{\sqrt{\pi^i}}{\Gamma(i/2+1)} \cdot d_m^i$$

$$P_{h,intersect} = min\{\tau, 1\}$$

where d_m is estimated by Equation (3) and σ_h can given by Equation (5).

Proof:

See Fig.6. The rectangle is a node MBR in level h whose side length is σ_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.

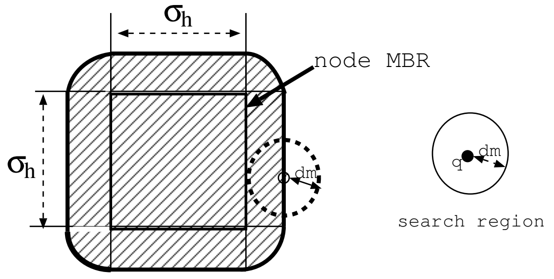

Figure 6: Example of Minkowski-sum in 2-dimensional space.

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,

$$P_{h,intersect} = P_{q,mink} = Vol_{mink} \quad (see \;\; Proposition \; 4)$$

where $P_{q,mink}$ refers to the the probability that q is contained in the Minkowski-sum; Vol_{mink} is the volume of Minkowski-sum. Thus, calculating the volume of Minkowski-sum in d-dimensional space is necessary for calculating $P_{h,intersect}$.

The following formula for calculating the volume of Minkowski-sum in d-dimensional space has been used in [2] and has been mathematically proved in [10].

$$Vol_{mink} = \sum_{i=0}^{d} Q_i \cdot \left(\begin{array}{c} d \\ i \end{array} \right) \sigma_h^{d-i}; \qquad Q_i = \frac{\sqrt{\pi^i}}{\Gamma(i/2+1)} \cdot d_m^i \tag{7}$$

In Equation (7), if Q_i is substituted into Vol_{mink} and $P_{h,intersect} \leq 1$ is considered, Proposition 5 can be proved. □

Lemma 1 *The expected number of nodes in level h that intersect with the search region, $N_{h,inter}$, can be given by*

$$N_{h,inter} = n_h \cdot P_{h,intersect} \tag{8}$$

where $P_{h,intersect}$ is estimated by Proposition 5 and n_h can be given by Equation (4).

Proof: To calculate $N_{h,inter}$ we have to sum $P_{h,intersect}$ for every node in this level. Considering the objects and the query point is uniformly distributed, all the nodes in this level have the same probability of intersecting with the search region. Thus, $N_{h,inter}$ can be given by multiplying $P_{h,intersect}$ 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), $Accesses_{node}$, is given by

$$Accesses_{node} = 1 + \sum_{h=1}^{H-1} N_{h,inter} \tag{9}$$

where H is the height of the R*-tree, $N_{h,inter}$ is given by Equation (8).

If the corresponding equations are substituted into Equation (9), the estimating formula of the node accesses can be given by Equation (10), where f_r, f_i, f_l and H will be discussed in Section 3.4.

3.4 Discussion of f_r, f_i, f_l and H

f_r, f_i and H in the above Equation (10) are discussed here.

Let F_i and 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 $F_l = 2 * F_i$ (for point objects) and $F_l = F_i$. Here the two cases are discussed separately.

$$\boxed{\begin{array}{l} Accesses_{node} = 1 + \sum_{h=1}^{H-1} f_r \cdot f_i^{h-1} \cdot min\left\{ \sum_{i=0}^{d} \left(\begin{array}{c} d \\ i \end{array} \right) \cdot \frac{\pi^{\frac{i}{2}} \cdot d_m^i \cdot \sigma_h^{d-i}}{\Gamma(i/2+1)}, \quad 1 \right\} \\ \text{where} \\ \quad d_m = \frac{1}{2\sqrt{\pi}} \sqrt[d]{\frac{\Gamma(d/2+1)}{n}} \cdot (\sqrt[d]{m} + \sqrt[d]{m-1}) \\ \quad \sigma_h = \frac{1}{2} \left(\frac{1}{n}\right)^{\frac{1}{d}} \left[\left(\frac{n}{f_r \cdot f_i^{h-1}} - 1 \right)^{\frac{1}{d}} + \left(\frac{n}{f_r \cdot f_i^{h-1}} \right)^{\frac{1}{d}} \right] \end{array}} \tag{10}$$

3.4.1 Case of $F_l = 2 * F_i$

According to the analysis made by C. Faloutsos and I. Kamel [6], the average node utilization of all nodes in R*-tree is 70%. Clearly,

$$\begin{aligned} n &= (0.7 * F_i)^{(H-1)} \cdot f_l = (0.7 * F_i)^{(H-1)} \cdot (0.7 * F_l) \\ &= 2 * (0.7 * F_i)^H \end{aligned}$$

$$H = \lceil log_{(0.7*F_i)}(n/2) \rceil \tag{11}$$

where F_i is given by user and it decides the size of each node.

As pointed in [3], the number of entries in the root node has its specific characteristics. Anyway, the number of the non-root nodes far exceed that of the root node (only one root node). Thus, we can think the average node utilization of the non-root nodes is also 70%. That is,

$$f_i = 0.7 * F_i \quad f_l = 0.7 * F_l = 2 * f_i \quad (F_l = 2 * F_i)$$

It is clear that

$$n = f_r \cdot f_i^{(H-2)} \cdot f_l = 2 f_r \cdot f_i^{(H-1)}$$

$$f_r = \frac{n}{2 f_i^{(H-1)}} \tag{12}$$

Now, we have to prove that f_r calculated by Equation (12) meets the limitation of $1 < f_r \leq F_i$.

According to Equation (11), $log_{f_i}(n/2) \leq H < log_{f_i}(n/2) + 1$. Then it becomes easy to prove $1 < f_r \leq F_i$, which is omitted because of the limitation of space.

3.4.2 Case of $F_l = F_i$ (say F)

In the same way as the former case, f_r, f_i, f_l and H can be given by

$$f_l = f_i = 0.7 * F \quad H = \lceil log_{f_i}(n) \rceil \quad f_r = \frac{n}{f_i^{(H-1)}}$$

Note that the above formulas in this paper 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.

4 Experimental Evaluation

Using uniformly distributed points we verified our estimating model as the three parameters (i.e., d, m and n) change. Since the analysis in this paper is based on the average case, the results are the average values of 100 random trials. Because of the space limitation, only the results of the case that $F_l = 2 * F_i$ (see Section 3.4) 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 do not change much.

4.1 Evaluation with Different d

Since the analysis in this paper is based on the average case, the results are the average values of 100 random trials.

Table 2 shows the calculated results and their experimental counterparts. Without loss of generality, we let m be 40 and n be 40,000.

Table 2: Evaluation as d increases (n=40,000, m=40).

d	$Fanout$	$Access_{node}$ calculated	tested	error rate (%)
2	40	5.97	5.80	2.85
4	20	36.80	35.46	3.64
6	13	239.56	229.52	4.19
8	10	1385.84	1260.38	9.05
10	8	4348.04	3673.34	15.52
12	7	5128.83	4751.60	7.36

From Table 2, 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 increases greatly as dimensionality grows, which is *dimensionality curse* of NN search mentioned in many works. We can understand this as follows.

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 leaving the query point as the dimensionality grows. Thus, the number of nodes that intersect with the search region will become larger. 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.

2. As dimensionality increases, the gap between calculated result and tested result gets larger. 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. Anyway, according to our experiments presented in Appendix 4.3, the error rate is reduced for large databases.

3. When d increases from 10 to 12, the error rate drops greatly. This is because the INN search tends to access all the nodes and all the objects in this case. Thus, for $Access_{node}$, both the calculated result and the tested result tend to the total number of nodes. And the error rate of $Access_{node}$ when $d = 12$ mainly comes from the difference between the calculated total number of nodes and the actual total number of nodes.

4.2 Evaluation with Different m

The results are shown in Table 3. The experiments were performed with $d = 4$, $n = 40,000$ and $Fanout$=20.

Table 3: Evaluation as m grows (d=4, n=40,000, $Fanout$=20).

	$Access_{node}$		
m	calculated	tested	error rate (%)
20	27.27	26.04	4.51
40	36.80	35.46	3.64
60	44.45	42.64	4.07
80	51.14	49.41	3.38
100	57.21	54.83	4.15

From Table 3, we know that the change of m has not much influence on accuracy of our model when m is relatively very small to n. Another observation is that performance of the INN search algorithm degrades as m increases. We think this is easy to understand.

4.3 Evaluation As n Increases

We changed our database cardinality from 200 to 200,000. The results are shown in Table 4. The experiments were performed with $d = 4$, $m = 40$ and $Fanout$=20.

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

	$Access_{node}$		
n	calculated	tested	error rate (%)
200	8.14	7.21	11.43
2000	30.77	28.53	7.28
20000	36.60	35.21	3.80
40000	36.80	35.46	3.64
60000	36.81	35.62	3.23
80000	37.84	36.81	2.72
100000	38.10	37.30	2.10
200000	38.82	38.13	1.78

From Table 4, we observe that

1. The error rate of our model 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 n increases. We think this is because that there exist the following two contrary factors that counteract each other to some extent.

   (a) On the one hand, as n grows, the point density increases. Thus, the d_m tends to become shorter and the volume of the search region becomes smaller.

   (b) On the other hand, as the point density increases, the node MBRs become smaller and density of nodes increases.

The issue occurs that how $Fanout$s are determined in our experiments and how does it affects on the error rate of our model. We choose $Fanout$s in the experiments such that the product of $Fanout$ and d keeps roughly fixed, 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. The reason that the fanouts in our experiments are set to small numbers is that, we think the experiment result also reflects the manner of the cases of larger datasets in this way.

5 Conclusion

In this paper we proposed a performance model to mathematically analyze performance of the INN search algorithm with m (the number of neighbor objects reported finally), n (database cardinality) and d (dimensionality) as parameters. In this model, the two key factors of the distance from the 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.

References

[1] G.R. Hjaltason, H. Samet. "Distance Browsing in Spatial Database". ACM Transactions on Database Systems, Vol. 24, No. 2, pages 265-318, June 1999.

[2] S. Berchtold, C. Bohm, D. A. Keim, HP. Kriegel. "A Cost Model For Nearest Neighbor Search in High-Dimensional Data Space". in Proceedings of ACM PODS Symposium, pages 78-86, Tucson, Arizona, 1997.

[3] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger. "The R^*-tree: An Efficient and Robust Access Method for Points and Rectangles". In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 322-331, May 1990.

[4] N. Roussopoulos, S. Kelley, F. Vincent. "Nearest Neighbor Queries". In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 71-79, San Jose, CA, June 1995.

[5] A. Papadopoulos, Y. Manolopoulos. "Performance of Nearest Neighbor Queries in R-trees". In Proceedings of International Conference on Database Theory, pages 394-408, Delphi, Greece, January 1997.

[6] C. Faloutsos, I. Kamel. "Beyond Uniformity and Independence: Analysis of R-trees Using the Concept of Fractal Dimension". In Proceedings of ACM PODS Symposium, pages 4-13, 1994.

[7] P. Boncz, S. Manegold, M. Kersten. "Database Architecture Optimized for the New Bottleneck: Memory Access". In Proceedings of the 25th International Conference on Very Large Data Base, pages 54-65, Edinburgh, Scotland, UK, September 1999.

[8] K. Kim, S. K. Cha, K. Kwon. "Optimizing Multidimensional Index Trees for Main Memory Access". In Proceedings of ACM SIGMOD International Conference on Management of Data, pages 139-150, Santa Barbara, California, USA, 2001.

[9] I. Kamel, C. Faloutsos. "On Packing R-trees". In Proceedings of the 2nd International Conference on Information and Knowledge Management, pages 490-499, Arlington, VA, November 1993.

[10] Changzhou Wang, Xiaoyang Sean Wang. "Indexing very high-dimensional sparse and quasi-sparse vectors for similarity searches". VLDB Journal 2001(9): 344-361.

YaoKai FENG received his B.E. degree and M.E. degree in Computer Engineering 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 reserch interests include multidimensional indexing, spatial databases, and XML databases. He is a member of IPSJ, IEEE and ACM.

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. Since 1990, he has been with the Graduate School of Information Science and Electrical Engineering, Kyushu University, Japan, where he is a professor. He is a member of IEICE, IPSJ, ACM, and IEEE.

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 reserch interests include spatial databases, and biomedical databases. He is a member of IPSJ, IEICE, ACM and IEEE.