Efficient Processing of k-Nearest Neighbor Queries in Spatial Databases with the SH-tree

DANG Tran Khanh, KÜNG Josef, WAGNER Roland
Institute for Applied Knowledge Processing (FAW)
University of Linz, Austria
Email: {khanh, jkueng, rwagner}@faw.uni-linz.ac.at

Abstract: *Nearest neighbor problem has special considerations among database researchers. In many cases of querying, users want to see returned results that contain database objects similar to a given query object, and especially these returned results are ranked according to their similarity to the query object. In this paper, we introduce two adapted algorithms to the SH-tree [KKW2001] from the state-of-the-art of corresponding research results for efficiently processing the nearest neighbor problem in spatial databases. We do intensive performance tests on synthetic data sets, which dimension number varies from 2 to 64, as well as on real data set. Our experimental results show that the SH-tree with these adapted algorithms totally outperforms the search performance of the SR-tree in both IO-cost and CPU-time when processing the k-nearest neighbor queries. This result also confirms our theory analyses in [KKW2001]: the SH-tree can efficiently scale to high dimensional spatial databases.*

Keywords: spatial databases, multidimensional access methods (MAM), bounding sphere (BS), minimum bounding rectangle (MBR), nearest neighbor (NN) query

1. Introduction

Efficient processing of nearest neighbor problem, which consists of k-nearest neighbor (k-NN) queries ($k \geq 1$), takes an important role in modern spatial database [1] applications as in multimedia databases [SeK1997], time-series databases [FRM1994], CAD/CAM systems [BeK1997], medical image databases [KSF+1996], Geographic Information Systems (GIS), astrophysics databases [RKV1995], etc.

The state-of-the-art of corresponding research results has been shown in [RKV1995] and [HjS1995]. The k-NN algorithm described in [RKV1995] is different from the latter in which the result object number k is known in advance. It implements an ordered depth-first traversal on a R-tree-based MAM. A sorted buffer of k current NNs is used to keep the result objects. During the descending phase, the algorithm computes MINDIST (i.e. minimum distance) between the query object and MBRs (minimum bounding rectangles) of each newly visited non-leaf node and sorts them into an Active Branch List (ABL). Then it applies pruning strategies to the ABL to remove

[1] In this paper, we also call spatial databases multidimensional databases.

unnecessary branches. The following work is iterated until the ABL is empty: For each iteration, the algorithm picks the next branch in the list and recursively applies itself to the node according to this branch. At a leaf node, the algorithm updates the result buffer if necessary concerning the distance of each object in the leaf to the query object and the distance of the current k^{th} NN in the buffer. The MBRs pruning is then done against the distance of this k^{th} NN in the result buffer. Conversely, the latter approach that is called incremental NN algorithm produces the result objects one by one; the result object number k is possibly unknown in advance. Specially, when the $k+1^{st}$ neighbor is obtained, the second approach does not need to be recomputed (k+1)-NN from scratch [HjS1999], [HjS1995]. This algorithm employs a priority queue but it traverses the index structure in best-first search manner. It means that when deciding which node to traverse next, the algorithm picks the node with the least distance in the set of all nodes have been visited [HjS1999]. This approach, as reported, significantly outperforms the k-NN algorithm in [RKV1995] for distance browsing queries in spatial databases that use the R-tree as an index structure. In our experiments with the adapted algorithms to the SH-tree (cf. section 3), we confirm this result for the IO-cost (accessed data page number) but not for the CPU time (cf. section 4).

Moreover, efficient processing of NN problems requires multidimensional access methods (MAM) [2] that make use of the proximity of objects to focus the search on only potential neighbors [RKV1995]. There are some surveys of MAMs that have been recently published as [KKW2001a], [GaG1998], [OOI1991] and so on. Because of the space limitation, we do not present an overview on MAMs here. Figure 1 depicts the evolution schema of MAMs in recent years, say from 1996 to 2001 [3] and they are categorized into four classes: the KD-tree based techniques; the R-tree based techniques; the hybrid techniques of both KD-tree and R-tree; and other techniques that cannot be classified into three above classes. More details are referred to [KKW2001a].

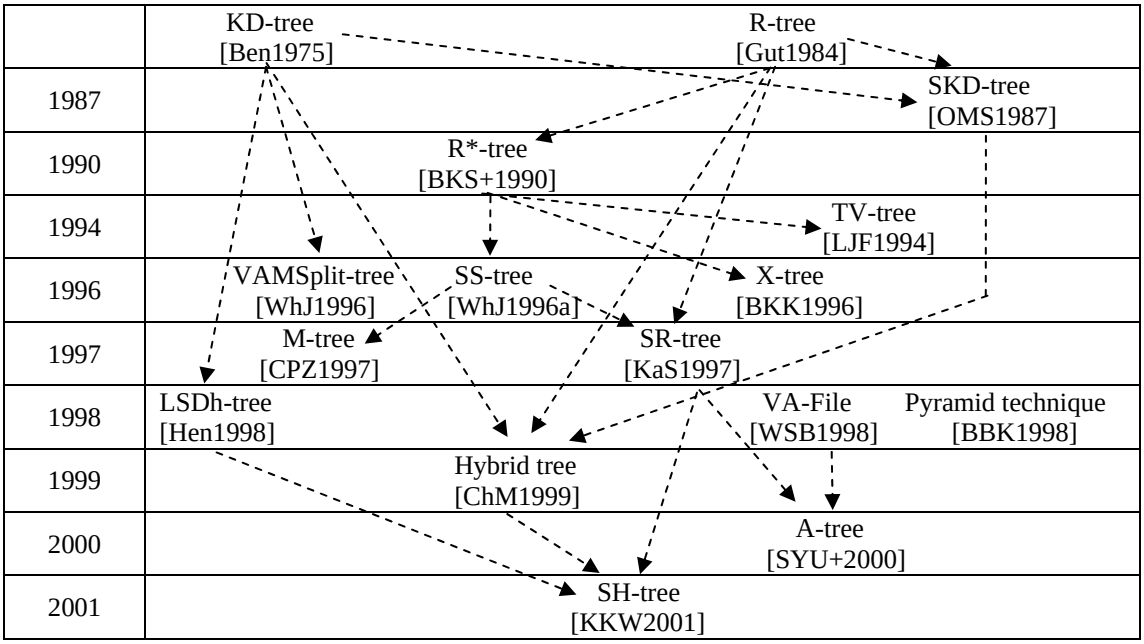


Figure 1: Evolution of multidimensional access methods (MAMs) in recent years

Some recent MAMs employ the algorithm in [RKV1995], typically as the X-tree [BKK1996], the SR-tree [KaS1997], the Hybrid tree [ChM1999], etc. The LSDh-tree [Hen1998] solves the NN

[2] They are also called spatial access methods (SAM).
[3] See [GaG1998] for an evolution schema of MAMs from 1966 to 1996

problem by using algorithm introduced in [Hen1994]. This algorithm, however, is similar to one published in [HjS1995] [4]. The A-tree [SYU+2000], as declared, improves the algorithm introduced in [HjS1995] but at the end, it results in using two shorted list during processing the queries: NNOL (Nearest Neighbor Object List) and NNVL (Nearest Neighbor VBR [5] List). This approach is the same as that of [Hen1994].

This paper is organized as follows: Section 2 reviews the SH-tree structure. Section 3 presents algorithms that are originated from two state-of-the-art algorithms but adapted to the special structure of the SH-tree for solving the k-NN queries. Section 4 gives experimental results with the implementation of these algorithms. The conclusions and future work are given in section 5.

2. The SH-tree

The SH-tree (Super Hybrid tree) [KKW2001] is a hybrid index structure formed by combining both of the KD-tree-based and the SR-tree-based index techniques. Originally, the SH-tree has been designed for not only point objects, but also for extended objects as polygons, lines, etc. Our current implementation of the SH-tree, however, is just for controlling point objects. The SH-tree, in fact, is motivated by alleviating the fan-out problem of the SR-tree as well as employing no overlap-free aspect between subspaces, which has been introduced in the SKD-tree [OMS1987] and the Hybrid tree [ChM1999], when partitioning data space.

The SR-tree was introduced by Katayama et al. and it has shown superiorities over the R*-tree [BKS+1990] and the SS-tree [WhJ1996a] by dividing feature space into both small volume regions (using bounding rectangles – BRs) and short diameter regions (using bounding spheres – BSs). Nevertheless, the SR-tree must incur the fan-out problem: only one third of the SS-tree and two third of the R*-tree (see [KaS1997]). The low fan-out causes the SR-tree-based searches to read more nodes and to reduce the query performance. However, the fan-out problem does not occur in case of the KD-tree based index techniques: it is constant for arbitrary dimension number. Therefore, the SH-tree overcomes the fan-out problem by employing the KD-tree like structure for presentation of internal nodes while still keeping the SR-tree-like presentation for its balanced and leaf nodes (see figure 3) to take inherent advantages of the SR-tree.

Besides, to easily control data objects that cross a split-selected position, say extended objects, and to solve the storage utilization problem, the SH-tree chooses no overlap-free space partitioning. Due to the limitation of space, we do not detail more this work here; see [KKW2001] for the detail discussion.

There are three kinds of nodes in the SH-tree: Internal, balanced and leaf nodes. Each internal node i carries information sp_i=<d, lo, up>, where d is split dimension, lo represents the left (lower) boundary of the right (higher) partition, up represents the right (higher) boundary of the left (lower) partition. While up=lo means no overlap between partitions, up>lo indicates that partitions overlap. Let BR_i denote bounding rectangle of internal node i. The BR of its left child is defined as $BR_i \cap (d \leq up)$. Note that $\cap$ denotes geometric intersection. Similarity, the BR of its right child is defined as $BR_i \cap (d \geq lo)$. Each balanced node and each leaf node have the similar structures as those of the SR-tree and the SS-tree, individually [KKW2001].

[4] These two algorithms are nearly published at the same time [HjS1999]
[5] VBR stands for Virtual Bounding Rectangle (see [SYU+2000] for the details).

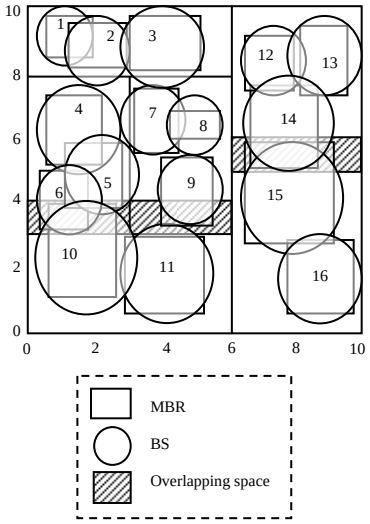


Figure 2: A possible partition of data space for a SH-tree

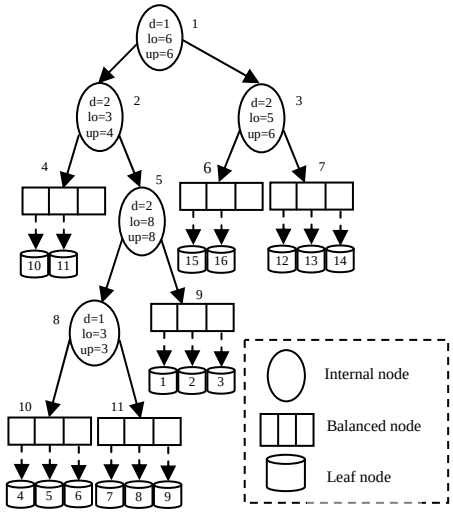


Figure 3: Mapping of space decomposition in figure 2 to the SH-tree

Figure 2 shows a possible partition of a 2-dimensional space and figure 3 shows corresponding mapping to the SH-tree. Assume we have a 2-dimensional feature space D with size of (0,0,10,10). As presented above, we have BR_1=(0,0,10,10), BR_2=$BR_1 \cap (d \le 6)$=(0,0,6,10), BR_3=$BR_1 \cap (d \ge 6)$ =(6,0,10,10), BR_4=$BR_2 \cap (d \le 4)$=(0,0,6,4), BR_5=$BR_2 \cap (d \ge 3)$=(0,3,6,10) and so on. This information of the bounding regions is not stored in the tree, but computed when it is needed. Operations on the SH-tree and related topics have been presented in [KKW2001].

3. Adapted Algorithms to Solve the Nearest Neighbor Problem for the SH-tree

This section presents adaptation of the state-of-the-art algorithms to the SH-tree for processing the k-NN queries in spatial databases. First, we introduce adapted k-NN algorithm that depends on the one of [RKV1995]. Figure 4 shows pseudo-code of the adapted algorithm.

Adapted algorithm 1:

```
1:  kNN_Query (Node, Obj, kNN_buffer, k, BR)
2:  NODE                Node          //Current node
3:  DATAOBJECT          Obj           //Query object
4:  OBJECTLIST          kNN_buffer    //Buffer of result objects
5:  INT                 k             //Length of kNN_buffer
6:  BOUNDINGREC         BR            //Bounding region of space according to Node

7:  Case (Node.type) Of
8:  INTERNAL_NODE:
9:      //Compute BR for left and right child
10:     BOUNDINGREC     BRleft=BR ∩ (d ≤ Node.up)
11:     BOUNDINGREC     BRright=BR ∩ (d ≥ Node.lo)
12:     //MINDIST from Obj to BRleft and BRright
13:     FLOAT           leftdist=MINDIST(Obj, BRleft)
14:     FLOAT           rightdist=MINDIST(Obj, BRright)
15:     If leftdist<rightdist Then
16:         If (kNN_buffer.num<k) or (kNN_buffer[k].dist>leftdist) Then
17:             kNN_Query (Node.left, Obj, kNN_buffer, k, BRleft)
18:         else    Return
```

```
19:                     If (kNN_buffer.num<k) or (kNN_buffer[k].dist>rightdist) Then
20:                             kNN_Query (Node.right, Obj, kNN_buffer, k, BRright)
21:             else  //leftdist ≥ rightdist
22:                     If (kNN_buffer.num<k) or (kNN_buffer[k].dist>rightdist) Then
23:                             kNN_Query (Node.right, Obj, kNN_buffer, k, BRright)
24:                     else    Return
25:                     If (kNN_buffer.num<k) or (kNN_buffer[k].dist>leftdist) Then
26:                             kNN_Query (Node.left, Obj, kNN_buffer, k, BRleft)
27:   BALANCED_NODE:
28:             For each entry Leaf  Do
29:                     If (kNN_buffer.num<k) or (kNN_buffer[k].dist>Distance(Obj, Leaf)) Then
30:                             kNN_Query (Leaf, Obj, kNN_buffer, k, NULL)
31:   LEAF_NODE:
32:             For each object Object Do
33:                     FLOAT dist=ObjectDistance(Obj, Object)
34:                     If dist< kNN_buffer[k].dist Then
35:                             kNN_buffer[k]=Object
36:                             kNN_buffer.AscendingSort()
37:   EndCase
```

Figure 4: Pseudo-code of the adapted k-NN algorithm 1

The algorithm presented in figure 4 implements a depth-first search that is similar to one introduced in [RKV1995]. Nevertheless, depending on the special structure of the SH-tree, this adapted algorithm does not use the ABL-Active Branch List. At the internal nodes, the algorithm recursively call itself depending on the MINDIST from the query object *Obj* to its left child *leftdist* and its right child *rightdist*. At the balanced node, the leaf is loaded if the distance *Distance(Obj, Leaf)* (line 29) from *Obj* to *Leaf* is less than the current k^{th} nearest neighbor. Note that, the distance *Distance(Obj, Leaf)* is defined as the longer one between the minimum distances to the leaf's MBR and BS (Bounding Sphere) [KaS1997], [KKW2001]. The function *ObjectDistance(Obj, Object)* (line 33) is used to compute the distance between two objects, which is Euclidean metric (L2 metric) in our implementation. Because the ABL is not used, its maintenance costs are omitted, which includes generating and sorting costs of the ABL (see [RKV1995] for the detail description of the originated algorithm). This is also a reason that improves its CPU-cost.

Now we introduce adapted k-NN algorithm that depends on the one of [HjS1995] (and also [HjS1999]). The originated algorithm has been proven optimality in terms of the accessed page number [BBK+1997], which is confirmed by our experimental results (cf. section 4). Figure 5 shows pseudo-code of the adapted algorithm to the SH-tree. It also employs a priority queue *PQ* and the result objects are output one by one (line 25). There is, however, a difference: this adapted algorithm needs the bounding region (BR) of the space as an input parameter to compute MINDIST for the left and right children of internal nodes from the given query object. This is easy because information of the data space is stored in the root node during creating the SH-tree. Therefore, BRs of its descendants can be computed as discussed in section 2. Besides, the algorithm also employs the functions *Distance* and *ObjectDistance*, which are described above. Although this adapted algorithm results in the better IO-cost, it must incur the maintenance costs for the priority queue PQ so the CPU-cost maybe higher than the first adapted algorithm in certain cases.

Adapted algorithm 2:

```
1:  kNN_Query (Obj, PQ, BR)
2:  DATAOBJECT          Obj             //Query object
```

```
3:  PRIORITYQUEUE       PQ              //Priority queue
4:  BOUNDINGREC         BR              //Bounding region of the space

5:  PQ.Push (Root, Root.type, 0.0)
6:  While not PQ.IsEmpty() Do
7:      PRIORITYQUEUE       top=PQ.Top()
8:      Case top.type Of
9:      INTERNAL_NODE:
10:         //Compute BR for left and right child
11:         BOUNDINGREC     BRleft=BR ∩ (d ≤ top.up)
12:         BOUNDINGREC     BRright=BR ∩ (d ≥ top.lo)
13:         //MINDIST from Obj to BRleft and BRright
14:         FLOAT           leftdist=MINDIST(Obj, BRleft)
15:         FLOAT           rightdist=MINDIST(Obj, BRright)
16:         PQ.Push (top.left, top.left.type, leftdist)
17:         PQ.Push (top.right, top.right.type, rightdist)
18:     BALANCED_NODE:
19:         For each entry Leaf Do
20:             PQ.Push (Leaf, LEAF_NODE, Distance(Obj, Leaf))
21:     LEAF_NODE:
22:         For each object Object Do
23:             PQ.Push (Object, OBJTYPE, ObjectDistance(Obj, Object))
24:     OBJTYPE:
25:         Report Object as the next nearest neighbor object
26:     EndCase
```

Figure 5: Pseudo-code of the adapted k-NN algorithm 2

4. Experimental Results

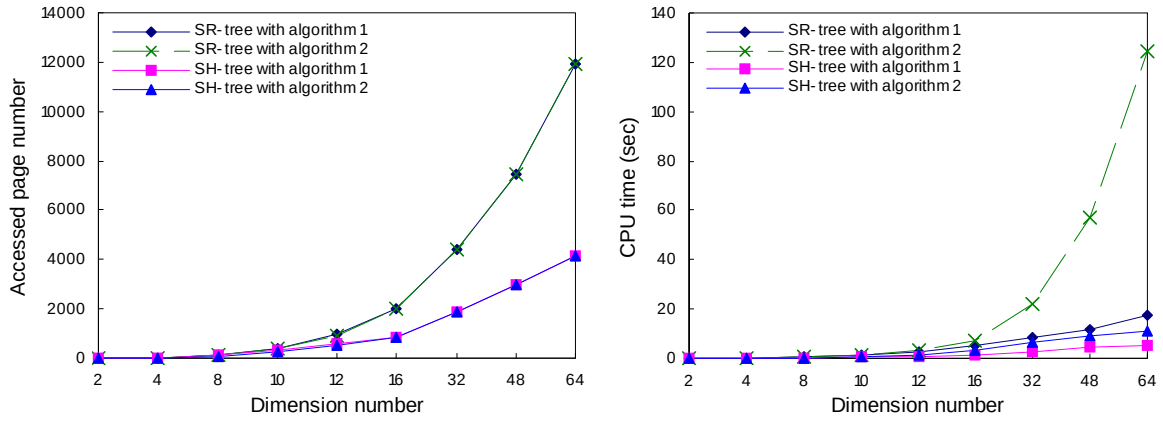
Our performance tests mainly concentrate on comparing performance of the SR-tree to that of the SH-tree with respect to processing the k-NN queries. We choose the SR-tree to do the tests because the SH-tree alleviates its fan-out problem as discussed and the SR-tree is also the one of most prominent index tree at the moment. Source code of the SR-tree is made available at *http://research.nii.ac.jp/~katayama/homepage/* by the authors. The search on the SR-tree are implemented in both of the originated algorithms, which are published in [RKV1995] and [HjS1995]. In the below charts, legends "*SH-tree with algorithm 1*" and "*SH-tree with algorithm 2*" are correspondent with each of them, individually.

For the tests, we use both uniformly distributed data sets [6] and real data set. Each uniformly distributed data set has 100000 tuples, which dimension number varies from 2 to 64. The real data set consists of 60000 9-dimensional image feature vectors (*http://kdd.ics.uci.edu/*), which are extracted from images and based on their color moments. All the tests are tackled on an ultra-sparc 300MHz; main memory is 256 Mbytes and some Gigabytes of hard disk capacity. All programs are implemented in C++. The page size is 8Kb for the data sets to meet with the disk block size of the operating system. For the uniformly distributed data sets, the minimum storage utilization factor is set to 40% for the leaf nodes and the reinsertion factor is set to 60%. For the real data set, these parameters are 30% and 50%, individually. For all experiments, we found that if the dimension number is less than or equal to 12, we do not use redistribution policy [KKW2001] between the leaf nodes during the insertion. In our experiments, this policy is employed for uniformly distributed data sets, whose dimension number is greater 12 and in these cases, the reinsertion factor is again

[6] Uniformly distributed data set is sometimes called synthetic data set

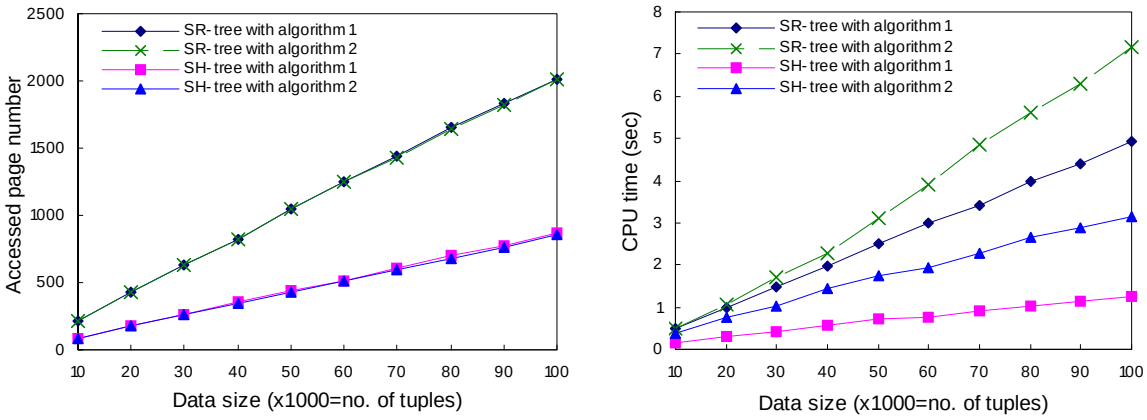
set to 30% as proposed in [BKS+1990]. Moreover, we do the tests to find 15 nearest neighbors in all experiments. For each test, 100 query points are randomly selected among the corresponding data sets.

Figure 6 shows experiment results to evaluate the performance according to various dimension number of the uniformly distributed data sets. The charts indicate that the SH-tree totally outperforms the SR-tree with respect to both of the adapted algorithms for both CPU-time and IO-cost. The first algorithm (figure 4) shows better CPU-time over the second one (figure 5). Nevertheless, the accessed page number of the second algorithm is less. Figure 7 shows performance of the SH-tree with a variety in data size of the 16-dimensional uniformly distributed data set. The results in both figures 6, 7 partly confirm the conclusion, as declared in [HjS1999], that the second adapted algorithm outperforms the first one, but only in terms of the IO-cost in this case. It also indicates well-scaled possibility of the SH-tree concerning the variety of data size. Figure 8 gives the performance evaluation of the SH-tree and the SR-tree with a variety in data size of the 9-dimensional real data set. The experimental results also prove superiority of the SH-tree to the SR-tree. Specially, the second adapted algorithm shows the better result over the first one. This result again confirms the conclusion that has been presented in [BBK+1997] that the second algorithm is optimal in terms of the accessed page number.



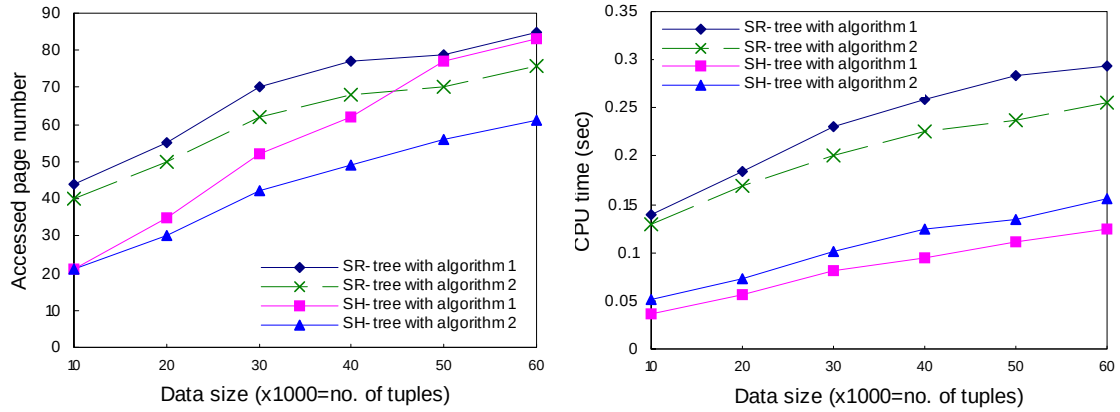
(a) (b)

Figure 6: Variety in dimension number of uniformly distributed data



(a) (b)

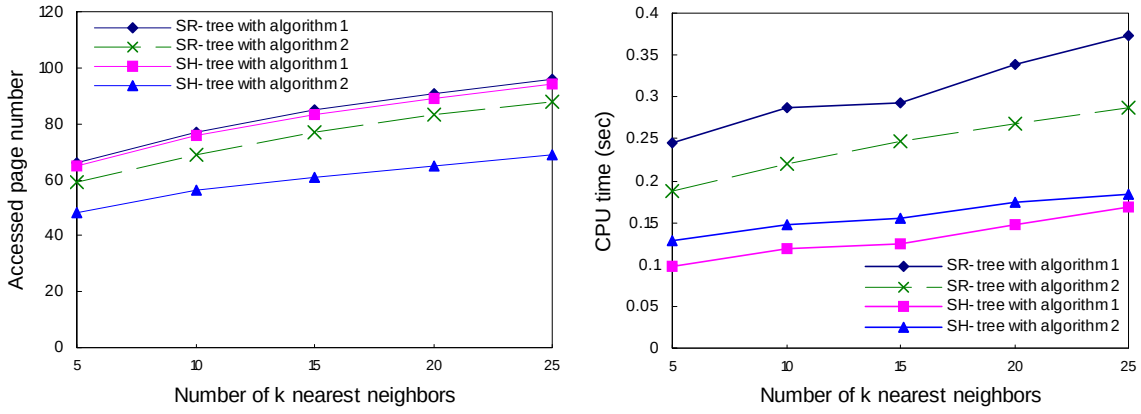
Figure 7: Variety in data size of 16-d uniformly distributed data



(a) (b)

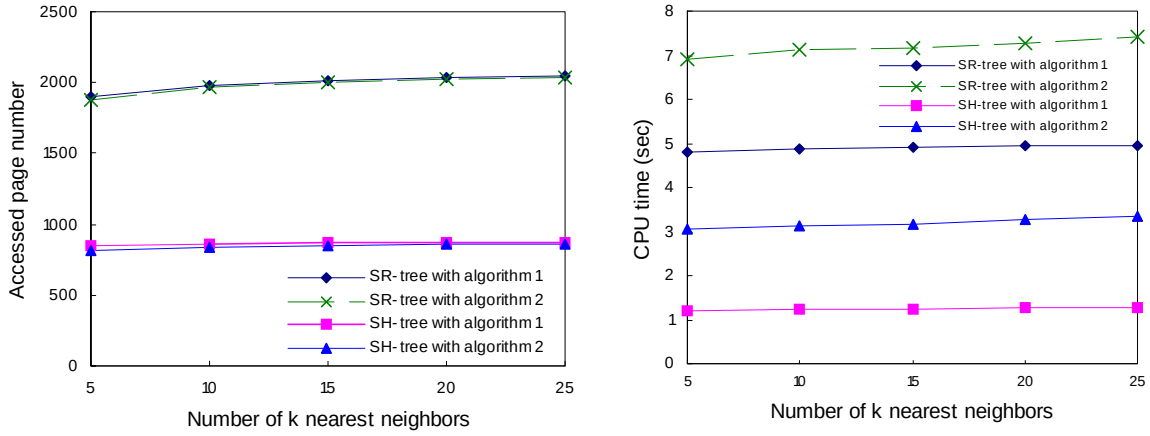
Figure 8: Variety in data size of 9-d real data

Figures 9 and 10 depict the performance of the trees concerning various number of k nearest neighbors for both real data set and 16-dimensional synthetic data set. The SH-tree also outperforms the SR-tree in all cases.



(a) (b)

Figure 9: Variety in number of NN (real data set)



(a) (b)

Figure 10: Variety in number of NN (16-d synthetic data

We can explain the experimental results as follows. In all the tests, the CPU-cost of the SH-tree is less than the one of the SR-tree because the algorithms do not compute the distance from the given query object to the bounding spheres at each internal node of the SH-tree, which must be done during the search of the SR-tree. Besides, the search based on the SH-tree accesses less data pages because of its special structure (see [KKW2001]). Our experiments confirmed this theory analysis. Nevertheless, the SH-tree just employs bounding regions at the internal nodes, so pruning when traversing maybe not suitable for very skewed data. In those cases, the IO-cost of the SH-tree can be higher than that of the SR-tree. Furthermore, operations at line 16,17,20 and 23 in figure 5 are expensive, it accounts for the higher CPU-time of the second algorithm as comparing to the one of the first algorithm (figure 4). These operations must ensure the priority queue PQ in ascending order. The same operation in figure 4 only occurs at line 36. In our test cases, although the IO-cost of the second algorithm is less than that of the first algorithm, it is not enough to absolutely compensate for the CPU-time as experimental results shown in [HjS1999]. However, the total elapsed time for the second algorithm with the real data set is better. For those reasons, selection of the efficient algorithms is also depended on size and distribution of data. The first adapted algorithm is very suitable for the SH-trees that fit in the memory because its CPU-cost is lower, while the second one is optimal in terms of IO-cost as shown in [BBK+1997].

5. Conclusions and Future Work

In this paper, we presented two adapted algorithms, which are originated from the state-of-the-art research results, to the SH-tree for efficiently processing the k-NN queries in the spatial/multidimensional databases. The experiments are conducted on both of the uniformly distributed data sets and the real data set. The results show that the SH-tree with these adapted algorithms efficiently processes the k-NN queries and outperforms the SR-tree by an order of magnitude in all the experiments. Our experimental evaluations also confirm the related conclusions in the previous researches presented in [HjS1999], [BBK+1997] concerning the optimality in terms of the IO-cost of the algorithm presented in [HjS1995], [HjS1999]. Besides, these results also prove the correctness of our theory analyses in [KKW2001]: the SH-tree can efficiently scale to the high dimensional spatial databases. Our further research will focus on spatial join operation and complex multi-attribute queries for which these results give us a solid base concerning the indexing aspects.

References

[BBK+1997] S. BERCHTOLD, C. BÖHM, D. A. KEIM, H. P. KRIEGEL. A cost model for nearest neighbor search in high-dimensional data space. Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of Database Systems, 1997, pages 78-86.

[BBK1998] S. BERCHTOLD, C. BÖHM, H.P. KRIEGEL. The Pyramid Technique: Towards Breaking the Curse of Dimensionality. Proc of ACM SIGMOD International Conference on Management of Data, 1998

[BeK1997] S. BERCHTOLD, H.P. KRIEGEL. S3: Similarity Search in CAD Database Systems. Proceedings of ACM SIGMOD International Conference on Management of Data, 1997, pages 564–567

[Ben1975] J. L. BENLEY. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM, Vol. 18, No. 9, pp 509-517, September 1975

[BKK1996] S. BERCHTOLD, D.A. KEIM, H.P. KRIEGEL. The X-tree: An Index Structure for High-Dimensional Data. Proceedings of 22nd International Conference on Very Large Databases, September 1996

[BKS+1990] N. BECKMANN, H.P. KRIEGEL, R. SCHNEIDER, B. SEEGER. The R*-tree: an efficient and robust access method for points and rectangles. Proceedings of ACM SIGMOD International Conference on Management of Data, 1990

[ChM1999] K. CHAKRABARTI, S. MEHROTRA. The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. Proceedings of 15th International Conference on Data Engineering 1999. IEEE Computer Society

[CPZ1997] P. CIACCIA, M. PATELLA, P. ZEZULA. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. VLDB 1997

[FRM1994] C. FALOUTSOS, M. RANGANATHAN, Y. MANOLOPOULOS. Fast Subsequence Matching in Time-Series Databases. Proceedings of ACM SIGMOD International Conference on Management of Data, 1994

[GaG1998] V. GAEDE, O. GÜNTHER. Multidimensional Access Methods. ACM Computing Surveys, Vol. 30, No. 2, June 1998

[Gut1984] A. GUTTMAN. R-Trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of ACM SIGMOD Conference, 1984

[Hen1994] A. HENRICH. A distance Scan Algorithm for Spatial Access Structures. Proceedings of the 2nd ACM Workshop on Advances in Geographic Information Systems, pages 136-143, Gaithersburg, Maryland, December 1994

[Hen1998] A. HENRICH. The LSD/sup h/-tree: An Access Structure for Feature Vectors. Proceedings of 14th International Conference on Data Engineering, 1998. IEEE Computer Society

[HjS1995] G. R. HJALTASON, H. SAMET. Ranking in Spatial Databases. Advances in Spatial Databases - 4th Symposium, SSD'95, Lecture Notes in Computer Science 951, Springer-Verlag, Berlin, 1995

[HjS1999] G.R. HJALTASON, H. SAMET. Distance Browsing in Spatial Databases. ACM Transactions on Database Systems, Vol. 24, No. 2, June 1999

[KaS1997] N. KATAYAMA, S. SATOH. The SR-Tree: An Index Structure for High Dimensional Nearest Neighbor Queries. Proceedings of the ACM SIGMOD International Conference on Management of Data, 1997

[KKW2001] D. T. KHANH, J. KUENG, R. WAGNER. The SH-Tree: A Super Hybrid Index Structure for Multidimensional Data. The 12th International Conference on Database and Expert Systems Applications - DEXA 2001, Germany.

[KKW2001a] D. T. KHANH, J. KUENG, R. WAGNER. Multidimensional Access Methods: Important Factor for Current and Next Decade's Applications in Spatial Databases. The 9th Interdisciplinary Information Management Talks – IDIMT 2001, Czech Republic

[KSF+1996] F. KORN, N. SIDIROPOULOS, C. FALOUTSOS, E. SIEGEL, Z. PROTOPAPAS. Fast Nearest Neighbor Search in Medical Image Databases. VLDB 1996, pages 215-226

[LJF1994] K. I. LIN, H.V. JAGADISH, C. FALOUTSOS. The TV-Tree: An Index Structure for High-Dimensional Data. VLDB Journal, Vol. 3, No. 1, January 1994

[OMS1987] B.C. OOI, K.J. MCDONELL, R. SACKS-DAVIS. Spatial KD-tree: A Data Structure for Geographic Databases. In Proc. COMPSAC 87, Tokyo, Japan, pages 433-438

[OOI1991] B. C. OOI. Indexing in Spatial Databases. URL: http://www.comp.nus.edu.sg/~ooibc/papers.html

[RKV1995] N. ROUSSOPOULOS, S. KELLEY, F. VINCENT. Nearest Neighbor Queries. Proceedings of ACM SIGMOD International Conference on Management of Data, 1995

[SeK1997] THOMAS SEIDL, HANS-PETER KRIEGEL. Efficient User-Adaptable Similarity Search in Large Multimedia Databases. VLDB 1997, pages 506-515

[SYU+2000] Y. SAKURAI, M. YOSHIKAWA, S. UEMURA, H. KOJIMA. The A-Tree: An Index Structure for High-Dimensional Spaces Using Relative Approximation. Proceedings of the 26th VLDB 2000

[WhJ1996] D. WHITE, R. JAIN. Similarity Indexing: Algorithms and Performance. Proc. SPIE Vol.2670, San Diego, USA, May 1996

[WhJ1996a] D.A. WHITE, R. JAINN. Similarity Indexing with the SS-Tree. Proceedings of the 20th International Conference on Data Engineering, 1996. IEEE Computer Society

[WSB1998] R. WEBER, H. J. SCHEK, S. BLOTT. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. Proceedings of 24th International Conference on Very Large Data Bases, August 24-27, 1998