ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 166-170
doi: 10.18178/wcse.2017.06.029

A KSNA-Tree Algorithm for the Top-k Exact Keyword Search in Spatial Databases

Ye-In Chang [1], Chia-En Li [2]+and Kai-Ning Yang [3]

Department of Computer Science and Engineering National Sun Yat-sen University
Kaohsiung, Taiwan, Republic of China

Abstract. In recent years, many websites and applications allow users to find objects which match with all of the query keywords and are close to a specified location. Tao *et. al.* propose a data access structure called the SI-index which integrates the inverted index with R-tree. However, it takes a long time for dealing with data of objects and some extra time for data decompression. Therefore, in this paper, we propose a KSNA-tree algorithm. The KSNA-tree integrates a spatial index NA-tree with inverted index. The contributions of our approach are as follows. First, our approach only construct one KSNA-tree, instead of building *n* R-trees for *n* keywords in the database. Second, we organize the data of objects according to their spatial number. This will avoid random access in the query processing by directly accessing the spatial number of a node. Third, we enhance each node in the KSNA-tree with the inverted index. We can prune a node and all of its child nodes immediately once we know that one of the query keywords is definitely not in the node. From our simulation results, we show that our proposed approach is more efficient than the SI-index.

Keywords: data mining, inverted index, keyword matching, spatial index structure, spatial database, top-*k*

1. Introduction

The geographic information system (GIS) is now widely used in positioning and finding objects. Many websites and applications require the abilities to find objects in a region or close to a specified location. An object in spatial database consists of two parts: (1) spatial information and (2) textual information. For instance, a user wants to find the 'Snoopy hotel' near Kaohsiung city. The 'Snoopy hotel' has two keywords in the keyword set and Kaohsiung city is the specified area. In this case, the top-*k* spatial keyword query aims at finding the objects that match with all keywords and are close to a specified location [1]. Tao *et. al.* propose a data access structure called the SI-index which integrates the inverted index with R-tree [2]. They use data compression approaches, gap-keeping and Z-value [3], for reducing the size of the SI-index. However, there are problems in their data structure. First, a large number of R-trees are built for storing data of objects. For *n* keywords, *n* R-trees must be constructed. It takes long time for dealing with data of objects and some extra time for data decompression. When data objects are updated/deleted/queried, their algorithm must traverse all the R-trees of the query keywords. Second, they partition the space to construct the spatial index. This will result in random access in the query processing. Third, there is no principle to decide which algorithm of SI-index would be applied to answer the top-*k* spatial keyword query under an unpredictable dataset. Therefore, in this paper, we propose the Keyword-Search-Nine-Area-tree (KSNA-tree) algorithm to process the top-*k* spatial keyword query. As compared to SI-index, we have only one KSNA-tree for all of the keywords. The rest of the thesis is organized as follows. Section 2 gives a survey of some methods for top-*k* spatial keyword query from the spatial databases. Section 3 presents the proposed algorithm. Section 4, we study the performance. Finally, we give a conclusion.

+ Corresponding author. Tel.: +886-7-5254350.
E-mail address: lice@db.cse.nsysu.edu.tw.

2. Related Work

To deal with the spatial keyword query processing, there are two primary issues: spatial index structure and keyword matching methods. Querying a spatial database is not like querying in relational databases. There is no total ordering of objects in spatial objects, so the traditional access methods used in relational databases cannot be directly apply to spatial databases [4]. The problem of top-*k* spatial keyword query is to find objects which match with all of the keywords and are close to a specified location. The straightforward way is to fetch all the objects which match with all of the keywords first and then browses all matched objects in ascending order of their distances to the query point *q*. But this may cost too much time. Therefore, Felipe *et. al.* proposed the IR^2-tree integrating R-tree with the signature file [1]. The R-tree partitions objects into regions (blocks), and signature file is directly embedded in those blocks. Zhang *et. al.* proposed the IL-Quadtree integrating inverted index with the quadtree [5]. The quadtree recursively partitions objects into four regions, and inverted index stores those points and regions which the keyword appears. On the other hand, Tao *et. al.* proposed the SI-index integrating inverted index with R-tree [2]. Differently from the IL-Quadtree, they provide two query processing algorithms: (1) *SI-b* algorithm and (2) *SI-m* algorithm. The *SI-b* algorithm performs queries by browsing the R-trees and the *SI-m* algorithm performs queries by directly merging the inverted index. These algorithms mentioned above focus on the problem of finding *k*-nearest neighbors and each of these *k* objects must contain the query keywords. Though some users may not know how to exactly spell the keyword, there are solutions like LBAK-tree [6] and NAAK-tree [7] to solve the approximate keyword queries problem.

3. The Keyword-Search-Nine-Area-tree Approach

In this section, we propose a Keyword-Search-Nine-Area-tree (KSNA-tree) as the spatial index, and use the inverted index as the keyword matching approach. The KSNA-tree augments a tree-based spatial index NA-tree [8] with abilities for *k*-nearest neighbors search and keyword matching. In spatial databases, objects could be the point data or the region data. Therefore, we use the NA-tree which proposed by Chang *et. al.* [8] to index the spatial objects. The NA-tree costs lower than R-tree series index structure [9] [10] in query processing. Moreover, the NA-tree can deal with point data and region data [8].

3.1. The KSNA index structure

A spatial object can be a polygon, a rectangle, or other shapes. There is a common way to characterize the spatial object by using the Minimum Boundary Rectangle (MBR). The MBR is oriented parallel to the coordinate axes X and Y. As Figure 1-(a) shown, *L* is the lower left coordinate and *U* is the upper right coordinate of the object. They represent an object as *O*(*L*, *U*). In NA-tree [8], a non-leaf node can have nine, four, or two child nodes. On the other hand, the data of objects can only be stored in the external node. Figure 1-(b) shows the spatial database *SD*1 and the corresponding NA-tree structure is shown in Figure 2. The *bucket_capacity* denotes the maximum number of entries which can be stored in the leaf node.

(a) (b)

Fig. 1: Explain of the spatial object: (a) an object representation: *O*(*L*,*U*); (b) the spatial database *SD*1.

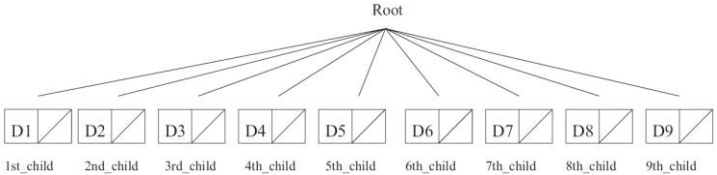
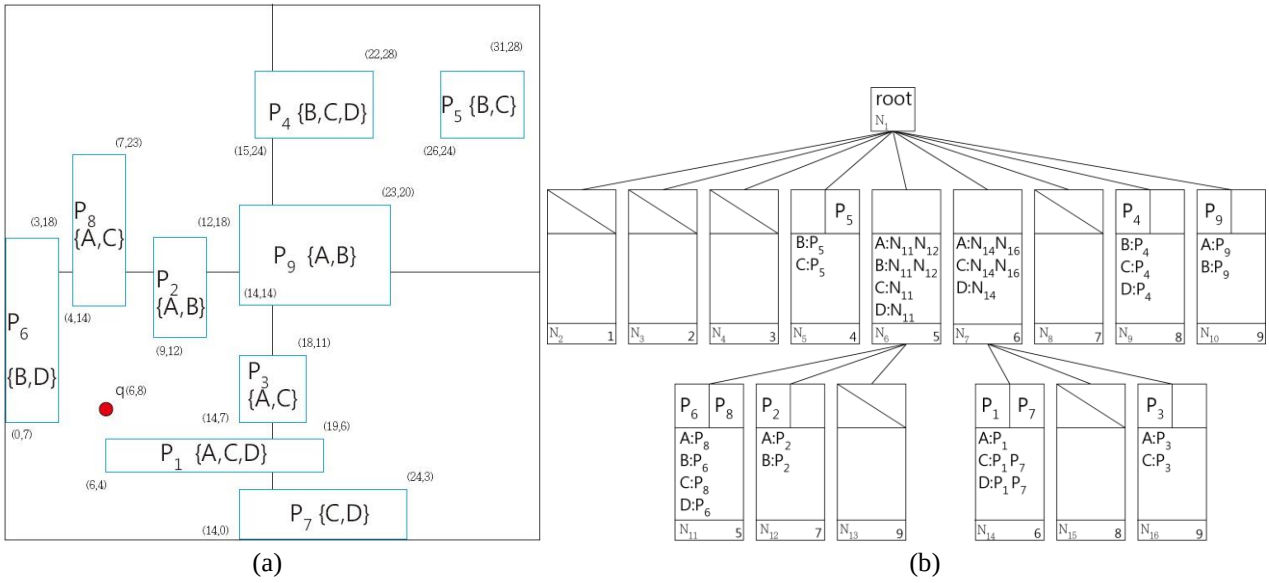


Fig. 2: The tree structure of the spatial database *SD*1 (*bucket_capacity* = 2)

Tao *et. al.* proposed the SI-index which combines the R-tree with the inverted index. For each keyword in the database, they both create a corresponding tree for storing objects which the keyword appears. They have a large number of trees for storing information of objects. It takes a long time for maintaining data and query processing. To solve this problem, we propose the KSNA index which combines an NA-tree with the inverted index. We locate spatial objects according to the spatial number, instead of partitioning the spatial space. Moreover, we enhance every node of KSNA-tree with inverted index. Once a keyword *w* does not match with a node *N*, no keyword of any child of node *N* will match with keyword *w*. Figure 3-(a) shows an example of the spatial objects and the corresponding KSNA-tree is shown in Figure 3-(b). Each non-leaf node stores the range of the spatial number and the inverted index of region. For instance, N_6 is a non-leaf node and the inverted index of node N_6 shows that node N_{11} and node N_{12} have the keyword *B*. On the other hand, each leaf node stores the geometric properties and the inverted index of objects. For instance, objects P_6 and P_8 are stored in the node N_{11} and the inverted index of the leaf node shows that only object P_6 has the keyword *B*.



(a) (b)

Fig. 3: Example: (a) a spatial dataset *SD*2; (b) the corresponding KSNA-tree for spatial dataset *SD*2.

3.2. Top-*k* spatial keyword query processing

We propose procedure *NA_Tree_TopK* to process the top-*k* spatial keyword queries. Figure 4-(a) shows our algorithm for processing top-*k* spatial keyword queries using the KSNA-tree. In query processing, a query point *Q* consists of two attributes: (1) the coordination *CQ* and (2) the textual condition *TC*. Procedure *NA_Tree_NN* is shown in Figure 4-(b). In procedure *NA_Tree_NN*, we repeatedly assign an element which is dequeued from the priority queue *PQ* to the next nearest object *NNO* and assign the distance which between the query point *Q* and the next nearest object *NNO* to the distance between query point and node (object) *DQN*. An example for the comparison between our proposed KSNA algorithm and two algorithms of SI-index: *SI-b* and *SI-m* is described as follows. In spatial dataset *SD*2, we want to find the top-1 nearest neighbor of the query point *q*(6,8) with keywords {B, C}. In the *SI-b* algorithm and the *SI-m* algorithm, they both construct 4 R-trees and have 20 nodes. Our KSNA algorithm only constructs 1 KSNA-tree and has 16 nodes. In the *SI-b* algorithm, the number of candidate nodes is 10, and the number of candidate objects is 8. On the other hand, in the *SI-m* algorithm, the number of candidate nodes is 6 and the number of candidate objects is 8. However, in our KSNA algorithm, the number of candidate nodes is 4 and the number of candidate object is 1. Therefore, our KSNA algorithm has better performance in terms of the processing time. We discuss the differences between our proposed KSNA-tree and the SI-index which is proposed by Tao *et. al*. First of all, the data structure for storing data is different. The SI-index uses R-tree for storing objects. The node of R-tree will be split into two nodes once the number of objects in the node reaches the capacity of node. Instead, we organize the data objects according to their spatial number. This will avoid random access in the query processing by directly accessing the spatial number of a node. Second, the number of trees is different. In SI-index, they build an R-tree and an inverted index for every keyword in the database.

As compared to SI-index, we have only one KSNA-tree for all keywords. In query processing, SI-index have to traverse more trees while the number of query keywords |W*q*| increases. In data maintaining processing, SI-index must traverse many trees to update or delete data of objects. Third, we apply inverted index in different way. Both of KSNA-tree and SI-index use inverted index for keyword matching. The result of query processing will be the intersection of all inverted indexes of query keywords. Besides, we enhance each node in the KSNA-tree with inverted index. Once we know that one of the query keywords is definitely not in the node, we can prune the node and the all child nodes of it immediately.

```
1: Procedure NA_Tree_TopK(RI, CQ, TC, NRO, AnsL);
2: Input:
3:   RI /* the Root of the Index */
4:   CQ /* Coordination of the Query */
5:   TC /* Textual Condition of the query, {w1, w2, ..., wn},
          each representing a keyword wi
6:   NRO /* Number of Retrieved Objects */
7: Output:
8:   AnsL /* the Answer List */
9: begin
10:   AnsL := ∅;
11:   PQ := ∅; /* Priority Queue */
11:   CNN := 0;
12:   PQ.Enqueue ((RI,0));
13:   while (CNN < NRO) do
14:   begin
15:     NAObject := NA_Tree_NN(CQ, TC, PQ, NNO);
16:     AnsL.add(NAObject);
17:     CNN := CNN + 1;
18:   end;
19: end;
```

(a)

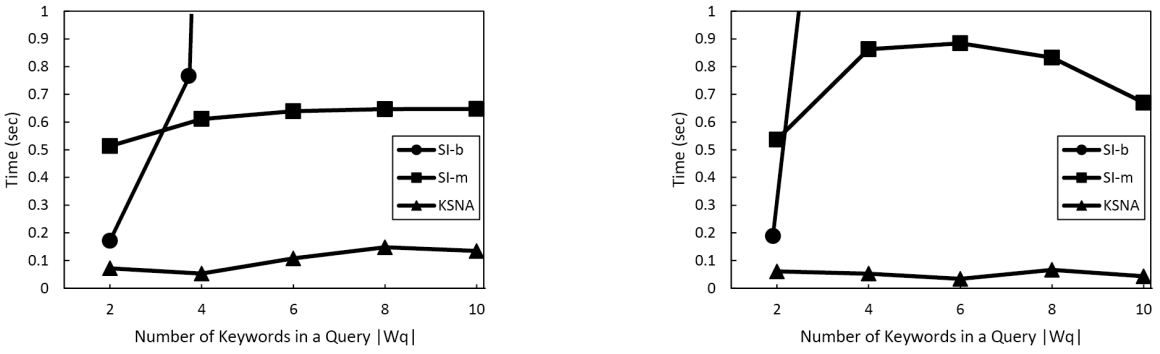
```
1: Procedure NA_Tree_NN(CQ, TC, PQ, NNO);
2: Input:
3:   CQ /* Coordination of the Query */
4:   TC /* Textual Condition of the query, {w1, w2, ..., wn} */
5:   PQ /* Priority Queue for NA-tree node*/
6: Output:
7:   NNO /* the Next Nearest Object */
8: begin
9:   while (PQ≠ ∅) do
10:  begin
11:    (NNO,DQN) := PQ.Dequeue();
12:    if (NNO is an non-leaf node) then
13:      for each entry TNode of NNO do
14:        if (TC match TNode's keywords) then
15:          PQ.Enqueue(TNode, MIN_Dist(CQ,TNode));
16:    else if /* NNO is a leaf node */
17:      for each entry NAObject of NNO do
18:        if (TC match NAObject's keywords) then
19:          PQ.Enqueue(NAObject, Dist(CQ,NAObject));
20:    else /* NNO is an object */
21:      return NNO as next nearest object to CQ;
22:  end;
23: end;
```

(b)

Fig. 4: Procedure: (a) *NA_Tree_TopK*; (b) *NA_Tree_NN*.

4. Performance

In this section, we study the performance of the top-*k* spatial keyword query processing using the SI-index and the KSNA-tree. In the database, there are 100000 objects and 500 different keywords. The dimensionality is 2 and each axis consisting of integers from 0 to 10000 [2]. The average number of keywords in each object is 100. The average number of objects for each keyword is 20000. The synthetic database has two kinds of data distribution: (1) Uniform and (2) Skew. These two datasets are different in the distribution of objects. In the dataset *Uniform*, the locations of objects are uniformly distributed. Each object is randomly assigned with one hundred keywords. On the other hand, in the dataset Skew, the data space is partitioned into N zones according to the Z-value of zone. The number of objects in each zone is decided based on the *Zipf* distribution [2]. Figure 5-(a) and Figure 5-(b) shows the comparison of the processing time of the dataset *Uniform* and *Skew* under the different size of query keyword |W*q*|, respectively.



(a) (b)

Fig. 5: A comparison of the processing time (sec) under the different size of query keyword |W*q*| (a) the dataset *Uniform*; (b) the dataset *Skew*.

In the dataset *Uniform*, the size of query keyword |W*q*| is changed from 2 to 10. Since the times of random access which the *SI-b* algorithm performs increase while the size of query keyword |W*q*| increases, the processing time of the *SI-b* algorithm increases dramatically. However, the KSNA algorithm uses the inverted index of nodes for pruning, the nodes which are pruned increase while the size of query keyword |W*q*| increases. Hence, the KSNA algorithm always outperform the *SI-m* algorithm. In the dataset *Skew*, the distribution of objects follows the *Zipf* distribution [2] and there are correlations between the keywords of objects. Hence, it is difficult to reduce the size of intersection of all inverted indexes for the *SI-m* algorithm. The KSNA algorithm always outperform the *SI-m* algorithm.

5. Conclusions

In this paper, we have proposed a KSNA-tree approach which can efficiently answer the top-*k* spatial keyword query. Our approach only constructs one KSNA-tree, instead of building *n* R-trees for *n* keywords in the database. Moreover, we have studied the processing time of our KSNA algorithm and two algorithms of SI-index: *SI-b* and *SI-m*. From our simulation results, we show that our approach is more efficient than the SI-index.

6. Acknowledgements

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST-105-2221-E-110-084.

7. References

[1] I. De Felipe, V. Hristidis, and N. Rishe. Keyword Search on Spatial Databases. *Proc. of the 24th IEEE International Conference on Data Engineering*. 2008, pp. 656-665.

[2] Y. Tao, and C. Sheng. Fast nearest neighbor search with keywords. *IEEE Trans. on Knowl. and Data Eng*. 2014, **26**(4): 878-888.

[3] M. Bern, D. Eppstein, and S. H. Teng. Parallel construction of quadtrees and quality triangulations. *Int. J. of Comput.* Geom. *and Appl*. 1999, **9**(6): 517-532.

[4] B. Moon, H. Jagadish, C. Faloutsos, and J. Saltz. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Trans. on Knowl. and Data Eng*. 2001, 13(1): 124-141.

[5] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted Linear Quadtree: Efficient Top K Spatial Keyword Search. *Proc. of the* 29th *IEEE International Conference on Data Engineering*. 2013, pp. 901-912.

[6] S. Alsubaiee, A. Behm, and C. Li. Supporting Location-Based Approximate-Keyword Queries. *Proc. of the 18th* SIGSPATIAL *International Conference on Advances in Geographic Information Systems*. 2010, pp. 61-70.

[7] Y. I. Chang, Z. S. Chen, and Y. G. Liou. NAAK-Tree: An Index for Querying Spatial Approximate Keywords. *Proc. of the 2013 National Computer Symposium*. 2013, pp. 1-6.

[8] Y. I. Chang, C. H. Liao, and H. L. Chen. NA-Trees: A dynamic index for spatial data. *J. of Inf. Sci. and Eng.* 2003, **19**(1): 103-139.

[9] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. *Proc. of the 1984 ACM SIGMOD International Conference on Management of Data*. 1984, pp. 47-57.

[10] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. *Proc. of the 1990 ACM SIGMOD International Conference on Management of Data*. 1990, pp. 322-331.