NAAK-Tree: An Index for Querying Spatial Approximate Keywords

Ye-In Chang, Zih-Siang Chen and Yan-Guo Liou
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
E-mail: changyi@cse.nsysu.edu.tw

***Abstract*—In recent years, the geographic information system (GIS) databases develop quickly and play a significant role in many applications. Many of these applications allow users to find objects with keywords and spatial information at the same time. Most researches in the spatial keyword queries only consider the exact match between the database and query with the textual information. Since users may not know how to spell the exact keyword, they make a query with the approximate-keyword, instead of the exact keyword. Therefore, how to process the approximate-keyword query in the spatial database becomes an important research topic. Alsubaiee $et\ al.$ have proposed the Location-Based-Approximate-Keyword-tree (LBAK-tree) index structure which is to augment a tree-based spatial index with approximate-string indexes such as a gram-based index. However, the LBAK-tree index structure is the R*-tree based index structure. It will spend more time when build the index and answer the spatial approximate-keyword query. Therefore, in this paper, we propose the Nine-Area-Approximate-Keyword-tree (NAAK-tree) index structure to process the spatial approximate-keyword query. From our simulation results, we show that the NAAK-tree is more efficient than the LBAK-tree to build the index and answer the spatial approximate-keyword query.**

***Index Terms*—Approximate-Keyword, Index Structure, Range Query, Spatial Database, Signature.**

I. Introduction

The publicly available geographic information system (GIS) databases contain important location-based information and play a significant role in many applications [7]. Due to the popularity of keyword search, particularly on the Internet, many of these applications allow the users to provide a list of keywords that the spatial objects should contain, in their description or other attribute [5]. The underlying location-based information entities present in GIS databases fundamentally comprise of two components: (a) spatial or location information and (b) textual information. The queries asked on location-based entities also contain spatial and textual components [7]. The queries are called *spatial keyword queries* [5, 7, 8].

Most existing researches in the spatial keyword queries only consider the exact match between the database and query with the textual information. However, users often do not know how to spell the exact keyword. They always make a query with the approximate-keyword. Therefore, the approximate keyword query processing in the spatial database becomes an important research topic.

The definition of the spatial approximate-keyword (SAK) query has two parts: the spatial query Q_s and the string query Q_t. A spatial condition Q_s is a rectangle or a circle. An approximate keyword condition Q_t has a set of k pairs $\{(w_1, \delta_1), (w_2, \delta_2), \ldots, (w_k, \delta_k)\}$, where each keyword w_i with an associated similarity threshold δ_i. The answer is to find all objects in the collection that are within the spatial region Q_s and satisfy the approximate keyword condition Q_t.

A *spatial access method* (SAM) is often designed based on the spatial index structure. It partitions the set of objects depending on the spatial proximity such that each group can fit into a disk page. Its structure depends on the distribution of objects in the embedded space, $e.g.$, the R-tree [6] and the R*-tree [3].

We can separate the domain of string matching into two parts, *exact matching* and *approximate string matching*. Exact matching means finding the positions of all *substrings* in the text which matches

the query pattern. The problem of approximate string match is to find the text positions that match a pattern with up to k errors. There are some approaches [10, 11], which use the inverted index and the q-grams to approximate string match.

The siqnature technique is that each attribute or keyword of a record yields a bit pattern of width W and K bits in W are set to 1. The disjunction of all these bit patterns is performed to form the record signature. There exists the fact that if a record satisfies a query, then every bit position set in a query signature must also be set in the record signature. It is possible that a record signature matches a query signature but the corresponding record does not satisfy the query. This is called a *false match*. The probability of false matches can be manage arbitrarily small by appropriate choice of parameters W and K [9].

Wang $et\ al.$ proposed the 3-level hybrid index structure integrating the R*-tree [3], inverted lists and q-grams of the keywords [11]. Yao $et\ al.$ proposed a structure called *MHR-tree* [12]. The MHR-tree is based on the R-tree [6] augmented with the min-wise signature and the linear hashing technique. Alsubaiee $et\ al.$ proposed the LBAK-tree index structure [1] which is to augment an R*-tree with approximate-string indexes such as a gram-based index. They store the keywords in the internal nodes. Base on these keywords in all the internal nodes, they separate these keywords into frequent keywords and non frequent keywords. And then, they use a function to decide which nodes to build approximate indexes.

However, the R*-tree index structure has to look up the MBRs of the children to decide which child to insert a new object. The children or objects have to be split and be reinserted in the R*-tree index structure when the node is overflow. Since that they store keywords in the internal nodes of the R*-tree, they can not deal with keywords when building the node of the R*-tree index structure at the same time, or it may take more time to deal with keywords when the internal nodes are split and reinserted.

When answering an SAK query based on the LBAK-tree index structure, they have to find out which nodes that satisfy the spatial condition of SAK query. It has to check all the space of the children in an internal node and process the textual condition with the answers. The approximate index can get the similar keywords sets corresponding to each keyword of the textual condition of the SAK query. However, based on their LBAK-tree index structure, they have to check all the similar keywords sets when processing in SA-Nodes and SK-Nodes even if there is one of the similar keywords set which is already an empty set. Therefore, finding out the intersects of similar keywords sets and the keywords stored in nodes takes more time when the level of nodes is higher (if the root is the top).

To avoid these problems, in this paper, we propose the Nine-Area-Approximate-Keyword-tree (NAAK-tree) index structure to process spatial approximate-keyword query. From our simulation results, we show that the NAAK-tree index structure is more efficient than the LBAK-tree structure.

The rest of the paper is organized as follows. In Section 2, we give a related work of some index structure for approximate keyword query. Section 3 presents the NAAK-tree index structure and the SAK query processing. In Section 4, we study the performance. Finally, we give a conclusion in Section 5.

II. The Related Work

The LBAK-Tree [1] has better performance than the previous approach [11,12] for approximate keyword query processing in the spatial database. In this section, we describe the data structure of the LBAK-Tree.

Alsubaiee $et\ al.$ proposed an index structure [1] which is to augment a tree-based spatial index with approximate-string indexes such as a gram-based index. The main idea of the LBAK-tree is to augment a tree-based spatial index with abilities for approximate string search and keyword search. They use the keyword-search ability to prune search paths. And the LBAK-tree which is an R*-tree that has been enhanced to support spatial approximate-keyword queries. They classify the LBAK-tree nodes into three types.

- **S-Nodes** : do not store any textual information such as keywords or approximate indexes, and can only be used for pruning based on the spatial condition [1].
- **SA-Nodes** : store the union of keywords of their subtree, and an approximate index on

those keywords. They use SA-Nodes to find similar keywords, and to prune subtrees with the spatial and approximate keyword conditions [1].

- **SK-Nodes** : store the union of keywords of their subtree, and prune with the spatial condition and its keywords. Note that they must have previously identified the relevant similar keywords once we reach an SK-Node [1].

III. The NAAK-Tree

In this section, we describe how to index the database by using the NAAK-tree. The NAAK-tree augments a tree-based spatial index NA-tree [4] with abilities for approximate string search and keyword search. We classifying the nodes of NAAK-tree into three categories, and query processing based on our index structure.

A. Data Structure

In this subsection, we first introduce the partition numbering scheme which is used to organize the spatial data in the NA-tree. Then, we introduce the data structure of the NA-tree [4]. Finally, we separate the nodes of the NA-tree into 3 types.

1) The Partition Numbering Scheme: There is a common way to characterize the spatial object by using the minimum boundary rectangle. The minimum boundary rectangle is oriented parallel to the coordinate axes, say *X* and *Y*. Thus an object *O* is represented by its four bounding coordinates, X_l, X_r ($i.e.$ the leftmost and rightmost *X* coordinates, respectively), Y_b, and Y_t ($i.e.$ the bottommost and topmost *Y* coordinates, respectively). Chang $et.\ al.$ use the two points, $L(X_l,\ Y_b)$ and $U(X_r,\ Y_t)$, to represent a spatial object [4], where L and U is the lower left coordinate and the upper right coordinate of the object, respectively. They represent an object as $O(L,\ U)$. L is same as U if the object is a point data.

2) The NA-Tree: Tree structures handling multi-dimensional data are constructed with internal nodes and leaf nodes. In NA-tree [4], an internal node can have nine, four, or two children. Since a leaf has no children, leaves are terminal nodes. Data can only be stored in the terminal node, not in an internal node.

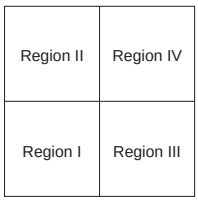

Fig. 1. The basic structure of NA-tree: four regions

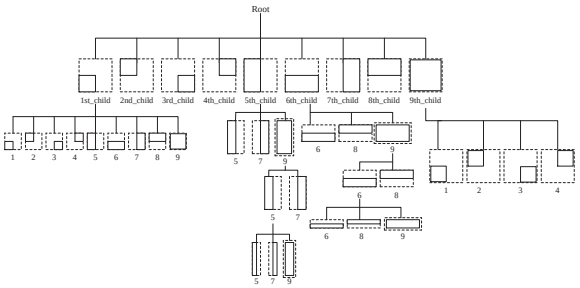

Fig. 2. The basic structure of an NA-tree

An NA-tree is a structure based on data location and organized by the spatial numbers. The spatial space is decomposed into four regions, as shown in Figure 1. Let *Region I* be the bucket numbers between 0 and $\frac{1}{4}(Max\_bucket + 1) - 1$, *Region II* be the bucket numbers between $\frac{1}{4}(Max\_bucket+1)$ and $\frac{1}{2}(Max\_bucket\ +\ 1)\ -\ 1$, *Region III* be the bucket numbers between $\frac{1}{2}(Max\_bucket + 1)$ and $\frac{3}{4}(Max\_bucket\ +\ 1)\ -\ 1$, and *Region IV* be the bucket numbers between $\frac{3}{4}(Max\_bucket + 1)$ and Max_bucket, where the Max_bucket is the maximal bucket number of the space. Based on this decomposition, an object is only nine cases are possible lying on the space, as shown in Figure 2.

3) Three Categories of Nodes: After indexing the spatial object by the NA-tree, we classify the nodes of the NA-Tree into three Categories:

- **FK(Frequent Keywords)-Nodes:** There are only the spatial condition in these nodes, but some FK-Nodes above an NFK-Node have the frequent keywords and approximate index on the frequent keywords.
- **NFK(Not Frequent Keywords)-Nodes:** There are the spatial condition, the union of keywords of their subtrees, and an approximate index on those keywords. The frequent keywords will be removed from the NFK-Nodes.
- **KS(Keyword Signature)-Nodes:** There are the spatial condition, the union of keywords of

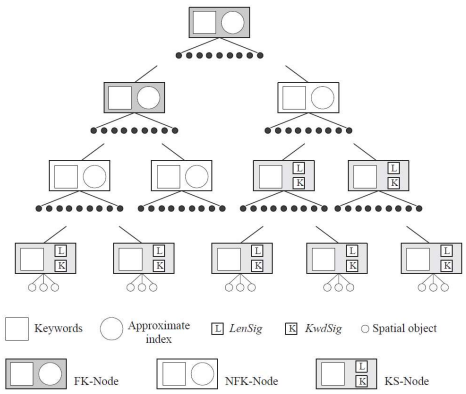

Fig. 3. The index structure model

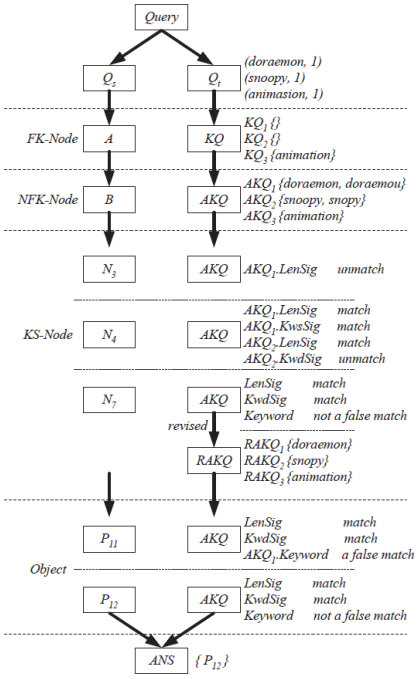

Fig. 4. The flowchart of the query

their subtrees, the $LenSig$ and the $KwdSig$ of the union of keywords of their subtrees.

After indexing the spatial condition and the textual condition, our index structure model is shown in Figure 3.

B. Spatial Approximate-Keyword Query Processing

In this subsection, we use a flowchart to describe how to processing SAK queries using the NAAK-Tree. Figure 4 shows the flowchart of the query processing.

Figure 5 shows an example with query $Q = \langle Q_s, \{(\text{doraemon}, 1), (\text{snopy}, 1), (\text{animasion}, 1)\}\rangle$.

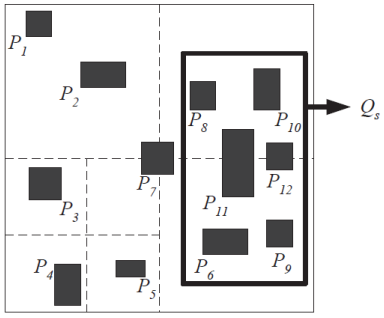

Fig. 5. A query example: the region related to Q_s.

Figure 5 is the region related to Q_s. The gray region also have objects but we choose some part of the map to process our algorithm. Table I shows the keywords in each object. There is a NAAK-tree built on the database, as shown in Figure 6. The objects in the database are all region data. Node A is an FK-Node which has an approximate index with frequent keyword 'animation'. Node B is a NFK-Node which has an approximate index with keywords from its children except the frequent keyword 'animation'. When query Q is issued, we initialize the similar-keyword sets KQ at root r. After processing root r, we find that the fourth child of the root intersect with Q_s and it is FK-Node A. Node A has the approximate index with frequent keyword 'animation', which is similar to (animasion, 1). Therefore, we can add 'animation' to KQ_3. After processing node A, we find that the third child of node A intersects with Q_s and it is NFK-Node B. We use the node B's approximate index to get keyword w_1's similar-keyword set {doraemon, doraemou} and keyword w_2's similar-keyword set {snoopy, snopy}, and add them to KQ_1 and KQ_2, respectively. At the end of processing node B, we get new similar-keyword sets and use AKQ_i to represent them, as shown in Table II. After processing node B, we find that the third, fourth, and seventh children of node B intersect with Q_s, push them into queue E. Therefore, we further check these children in the queue E with signature first. We represent these children by N_3, N_4, and N_7. At node N_3, the $LenRecSig$ and $KwdRecSig$ of node N_3 are the record signature of object P_6 and P_9, as shown in Table III-(a) and Table III-(b). We can ignore node N_3 immediately because $N_3.LenRecSig \cap AKQ_1.LenSig_i \neq AKQ_1.LenSig_i$, as shown in

TABLE I
THE KEYWORDS IN EACH OBJECT

P	Keywords
P_1	snoobe, kitti
P_2	snoopy, kitty, animation
P_3	snoobe, animation
P_4	doraemon, snoopy
P_5	doraemou, kity
P_6	kitty, winnie, animation
P_7	snoopy, kity
P_8	doraemou, snoobe, animation
P_9	snopy, mickey
P_{10}	winnie, doraemon
P_{11}	doraamou, snopy, animation
P_{12}	doraemon, snopy, animation

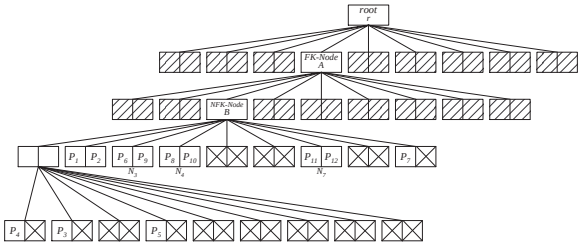

Fig. 6. The NAAK-tree of the example

Table III-(c). It is not matched, which means that the node N_3 does not have any keyword similar to keyword w_1(doraemon). According to the step that shown in Figure 4. Finally, the answer set is P_{12}.

IV. PERFORMANCE

In this section, we compare the performance of the NAAK-tree index structure with that of the LBAK-tree index structure in terms of the CPU time.

We conduct our experiment on the dataset that the data space is $10000 * 10000$. The simulation is performed with the following variable parameters settings as shown in Table IV. The threshold which is used to decide the frequent keyword sets

TABLE II
THE SIMILAR-KEYWORD SETS WITH SIGNATURE AND Q_t

AKQ	Similar keywords	LenSig	KwdSig
AKQ_1	doraemon	01000	100000000010000
	doraemou	01000	000000001010000
AKQ_2	snoopy	00010	000010000010000
	snopy	00001	000010000010000
AKQ_3	animation	10000	100000000000010

TABLE III
THE SIGNATURE AND KEYWORDS OF: (A) P_6 AND P_9; (B) N_3; (C) AKQ_i.

P	LenRecSig	KwdRecSig	Keywords
P_6	10011	100110100100010	kitty, winnie, animation
P_9	00011	010010000010000	snopy, mickey

(a)

N	LenRecSig	KwdRecSig	Keywords
N_3	10011	110111100110010	kitty, winnie, snopy, mickey, animation

(b)

AKQ	LenSig	KwdSig	Similar keywords
AKQ_1	01000	100000000010000	doraemon
	01000	000000001010000	doraemou
AKQ_2	00010	000010000010000	snoopy
	00001	000010000010000	snopy
AKQ_3	10000	100000000000010	animation

(c)

TABLE IV
PARAMETERS SETTING USED IN THE EXPERIMENT

Parameters	Description
TV	The threshold value of the node
NOB	The number of data objects
SB	The space budget for building approximate indexes

to 0.9.The size of length signature and keyword signature are set to 10 and 26, respectively. The q of q-grams in a gram-based approximate index is set to 2. For query processing, we compare the two methods using the average CPU time for processing 100 queries. The threshold value of the node TV is set to 20. And the space budget for building approximate indexes SB is set to 500MB.

Figure 7 shows the simulation result of the comparison of the construction of the LBAK-tree index structure and the NAAK-tree index structure. From Figure 7, we show that the performance of the NAAK-tree index structure is better than that of the LBAK-tree index structure.

Figure 8 shows the simulation result of the comparison of the execution time of SAK query in LBAK-tree index structure and the NAAK-tree index structure with different number of data objects *NOB*. From Figure 8, we show that the execution time of SAK query in LBAK-tree index structure increases faster than in the NAAK-tree index struc-

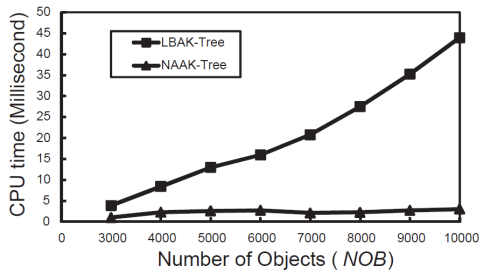

Fig. 7. A comparison of the execution time of building the index with different *NOB*

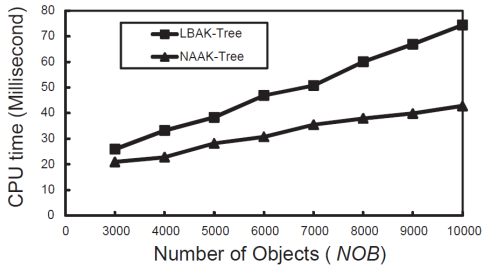

Fig. 8. A comparison of the execution time of SAK query with different *NOB*

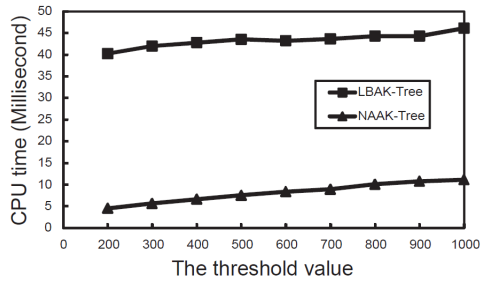

Fig. 9. A comparison of the execution time of building the index with different *TV*

ture.

In Figure 9, we set the space budget for building approximate indexes *SB* to 100MB, and the number of data objects *NOB* to 10000. It shows the simulation result of the comparison of the construction of the LBAK-tree index structure and the NAAK-tree index structure with different threshold values of leaf nodes. From the result, based on the NAAK-tree index structure, the performance of building the index with different *TV* is still better than the LBAK-tree index structure.

V. Conclusion

In recent years, with the developments of Web sites which support location-based keyword search, SAK queries become a practical and popular problem. This problem focuses on effectively building index and speeding up the SAK queries. In this paper, we have proposed an NAAK-tree index structure. From the simulation results, we have shown that building the index and answering SAK queries based on our proposed NAAK-tree index structure is more efficient than based on the LBAK-tree index structure.

A. Acknowledgements

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-95-2221-E-110-101.

References

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

[2] S. Alsubaiee and C. Li, "Fuzzy Keyword Search on Spatial Data," Proc. of the 15th Int. Conf. on Database Systems for Advanced Applications, pp. 464-467, 2010.

[3] N. Beckmann, H. P. Begel, R. Schneider, and B. Seeger, "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles," Proc. of the 1990 ACM SIGMOD Int. Conf. on Management of Data, pp. 322-331, 1990.

[4] Y. I. Chang, C. H. Liao, and H. L. Chen, "NA-Trees: A Dynamic Index for Spatial Data," Information Science and Engineering (SCI), Vol. 19, No. 1, pp. 103-139, 2003.

[5] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword Search on Spatial Databases," Proc. of the 24th Int. Conf. on Data Engineering, pp. 656-665, 2008.

[6] A. Guttman, "R-Trees: A Dynamic Index Structure for Spatial Searching," Proc. of the 1984 ACM SIGMOD Int. Conf. on Management of Data, pp. 47-57, 1984.

[7] R. Hariharan, B. Hore, C. Li, and S. Mehrotra, "Processing Spatial Keyword(SK) Queries in Geographic Information Retrieval (GIR) Systems," Proc. of the 19th Int. Conf. on Scientific and Statistical Database Management, p. 16, 2007.

[8] S. Jordy, F. Flavius, H. Frederik, and C. Vadim, "Semantic Web service discovery Using natural language processing techniques," Expert Systems with Applications, Vol. 40, No. 11, pp. 4660-4671, 2013.

[9] C. S. Roberts, "Partial-match Retrieval via the Method of Superimposed Codes," Proc. of the IEEE, Vol. 67, No. 12, pp. 1624-1641, 1979.

[10] D. M. WU, M. L. Yiu, and S. J. Christian,"Moving Spatial Keyword Queries: Formulation, Methods, and Analysis," ACM Transactions on Database Systems, Vol. 38, No. 1, pp. 1-47, 2013.

[11] Z. Wang, M. Du, X. Shi, and J. Le, "An Efficient Approach for Approximate Keyword Query in Geographic Information System," Proc. of the 2009 IEEE Int. Conf. on Intelligent Computing and Intelligent Systems, pp. 603-607, 2009.

[12] B. Yao, F. Li, M. Hadjieleftheriou, and K. Hou, "Approximate String Search in Spatial Databases," Proc. of the 2010 IEEE 26th Int. Conf. on Data Engineering, pp. 545-556, 2010.