Enhanced Baseline Algorithm for Best Keyword Search

Haribabu Valleti
Student, Department of CSE
KMM Institute of Technology and Sciences
Tirupati, India

S. Sivarama Krishna
Head of the Department, CSE
KMM Institute of Technology and Sciences
Tirupati, India

Abstract: **Temples, Universities, Airports, Colleges etc. represented as spatial database objects. Spatial database objects are mapped with keywords. Each keyword represents a service or business or process. Closest keyword search is called keyword cover, is most important and most popular problem in querying spatial database objects. Main objective of closest keywords search is to find spatial database objects that have minimum inter- objects distance for a given set of query keywords. Keyword rating is one of the best decision making techniques. A new closest keyword search technique is proposed, which considers inter object distance, query location and keyword rating of associated spatial database objects. Proposed new technique is scalable and generalized version of closest keywords search. New method is called scalable search for keyword cover.**

Keywords: spatial database, keyword search, keyword cover.

I.INTRODUCTION

Applications of spatial keyword search are increasing rapidly. Location-based services, satellite images, maps, vehicle movement details, details of computing powers, graphics facilities are some of the applications of keyword search problem. A spatial database object is represented by a tuple in the spatial database and each spatial database object is associated with one or more keywords. Each keyword specifies the function, service or finds spatial database objects that are associated with given set of query keywords. Only those spatial database objects that satisfy the following three goals are selected in the new proposed keyword cover problem.

1. Minimum inter object distance
2. Maximum keyword rating of objects
3. In a given location

More generally, every query is associated with many query keywords and every spatial database object is associated with one or more query keywords. For example, a business person wants to know temple darshan, hotel availability, airport distance, nearest bus stand and nearest university etc altogether to complete the visit with reference to a specific location. Business person actually wants to complete all these visits with minimum distance travelled and with minimum expenditure. No single database object is associated with all keywords. Each spatial database object is associated with a subset of all keywords and assure minimum inter object distance is called keyword cover. The problem of satisfying keyword cover property is called M closest keywords (MCK) problem j. MCK query requires to satisfy minimum inter object distance among the queried spatial data base objects associated with query keywords. Newly proposed method imposes two extra constraints.

1. Only those special database objects which are very close to the query location must be retrieved and
2. In addition to the minimum inter objet distance and query location keyword rating property must also be satisfied.

The new method is more generalized version of M Closest Keywords (MCK) problem and this new keyword cover problem is called Scalable Search for Keyword Cover (SSKC). SSKC considers minimum inter object distance, query location and query keyword rating parameters. In many business services keyword rating plays an important role in purchases. Temples, universities, colleges, hotels and many more services are rated by many customers from all over the world. For example, a university is rated 96 out of 100, a temple is rated 46 out of 60, a hotel is rated 6.5 out of 10. Now a day's customer buying decisions are directly influenced by the online business rating value. Many researchers have been carried out to find trends online purchases. They show that keyword rating occupies in first place in decision making.

R-tree is m-way and multilevel indexing data structure for indexing the spatial database objects. R*-tree is a variation of R-tree. New method consist two algorithms, base line algorithm and scalable keyword nearest neighbor algorithm. In the baseline algorithm top hierarchical levels of KRR*-trees are combined in order to generate potential candidate keyword covers. Best candidate solution is obtained based on some priority scheme after combining child nodes of already created nodes in the top level. The performance of baseline algorithm decreases rapidly as the number of query keywords increases. Drastic reduction in performance of baseline algorithm occurs because very large candidate keyword covers are generated. Definitely this is the main disadvantage of baseline algorithm.

Authors have proposed a new algorithm called scalable keyword nearest neighbor algorithm. This algorithm is very much scalable for larger sets of query keywords because it uses a very clever technique in generating candidate keyword covers. Keyword nearest neighbor algorithm selects one query keyword as principle query keyword. All the objects that are associated with the principle query keyword are called principal objects. For each principal object Local Best Keyword Cover (LBKC) is computed. Out

of all these LKBC details, the one with the highest evaluation is the solution for the keyword nearest neighbor method. Generalized candidate keyword covers are very much less when compared keyword nearest neighbor algorithm with that of baseline algorithm. Further processing of keyword candidate covers that are generated in the next level is optimal and very less in keyword nearest neighbor algorithm when compared with the baseline algorithm.

II. BACKGROUND

Each object in the spatial database is associated with one or a set of keywords. For simplicity, we assume that an object associated with multiple keywords is transformed to multiple objects located in the same location but associated exactly with only one keyword. In the spatial database an object details are represented. In the form of (identification no, x-value, y-value, keyword name, rating).Here x-value and y-value are object location details in the two dimensional geographical space. We assume that all query keywords correct and there are no spelling errors.

Assume that O = {o_1, o_2, o_3 …., o_n} be a set of objects. The diameter of "O" is defined as

$$\text{Diameter (O)} = \max_{o_i o_j \in O} \text{distance}(o_i, o_j) \quad \rightarrow (1)$$

For any two objects $o_{i,} o_{j \in}$ o, dist (o_i, o_j) is the Euclidean distance between $o_{i,}$ and $o_{j,}$ in the two-dimensional geographical space.

Based on the diameter of O and keyword rating of objects in O, the score of O is computed. Keyword ratings are assigned based on the interestingness of users. Generally users expect to maximize the minimum keyword rating of objects in keyword nearest neighbor. Sometimes users may be interested to maximize the weighted average of keyword ratings. A special function called linear interpolation function is applied in order to compute score of O. Both diameter and keyword rating are normalized and then score of is computed.

$$\text{O.score} = \text{score (A, B)} = \alpha\left(1 - \frac{A}{maxdist}\right) + (1\text{-}\alpha)\frac{B}{maxrating} \rightarrow (2)$$

A = diameter (O)

B = $min_{o \in O(o.rating)}$,

Where A is the diameter of the set of objects and B is the minimum keyword rating of objects in the set O and $\alpha(0 \leq \alpha \leq 1)$ is a parameter which application specific. If $\alpha = 1$ then the score of O is completely determined by the diameter of o. In this particular case keyword query. If α value is zero then the score of O is completely determined based on minimum keyword rating of the set O of objects only. In order to overcome the conflicts of different keyword ratings and different distances these two values are normalized such that the resulted value is in the closed interval [0, 1]. Here, max distance is taken as the maximum distance between any two objects in the entire spatial database D, and max rating is the maximum keyword rating of all the objects. Creating indexes for keyword ratings.

In the proposed method a new indexing tree structure called KRR*-tree used. KRR*-tree is an extended version of R*-tree with one extra added dimension to index keyword ratings. Keyword rating dimension and spatial database dimensions are completely different from each other. So, KRR*-tree is a three dimensional extended R*-tree for keyword rating. Once again for controlling conflicts among the measures keyword rating and spatial dimensions are normalized in the closed interval [0, 1]. Assume that D is a set of objects in the spatial database and we have to construct a KRR*-tree for the objects in D. Each object o∈D is mapped into a new three dimensional space by using the following function called mapping function.

$$f = o(x, y, rating) \rightarrow o\left(\frac{x}{max_x}, \frac{y}{max_y}, \frac{rating}{maxrating}\right) \rightarrow (3)$$

Where max_x, $max_{y,}$ max-rating are the maximum value of objects in D on x, y and keyword rating dimensions respectively. Actual procedure of constructing KRR*-tree is same as that of R*-tree construction in three dimensional space. Each node N in KRR*-tree is defined as 6-tuple, N(x, y, r, $l_x, l_y, l_{r)}$ where x is the value of N in x axis close to the origin i.e., (0,0,0,0,0,0) and l_x is the width of N in x axis y, l_y and r, l_r are similarly compute. The following diagram demonstrate how the nodes of KRR*-tree are indexed for the objects in keyword "hotel".

Generally a single tree is a constructed to index objects of different keywords and then the single tree is extended with an additional dimensional to index the keyword rating. Single tree index is best when most of the keywords are query keywords. In reality only a single tree index is poor to create spatial relationship. In the proposed method a single tree is used for each keyword. The KRR*-tree for the keyword k_i is denoted as KRR*k_i-tree.

III. FIRST BASELINE ALGORITHM

Baseline algorithm is really an extension of m-closest keywords (MCK) query method. In baseline algorithm design both top-down and bottom-up indexing techniques are used. Assume that there are n query keywords and n KRR*-trees have been constructed. That is there exist one to one mapping between keywords and KRR*-trees. Suppose T={k_1,k_2,k_3,… k_n} is the set of n query keyword. The child nodes of the root of KRR*-tree (1≤i≤n) are retrieved and they are combined in order to generate candidate keyword covers. Suppose for a given candidate keyword cover o = { $N_{k1,} N_{k2,} N_{k3,\ldots\ldots,} N_{kn}$ }, Where N_{ki} is a node of $KRR*_{ki}$-tree.

O.score = score (A, B)

A = $\max_{Ni,Nj \in o}$ $distance(N_{i,} N_j)$ →(4)

B = $\min_{N \in o}(N.maxrating)$

Where N.maxrating is the maximum value of objects under N in the keyword rating dimension and the value dist(N_i, N_j) is the minimum Euclidean distance between nodes N_i and N_j in the two dimensional geographical space represented by x and y dimensions.

Pseudo-code of the baseline algorithm is given in algorithm1. T is a set of query keywords. The baseline algorithm first generates candidate keyword covers after executing the function generate candidate. This function combines all the child nodes of all the root of all the KRR*k_i-trees for all the keywords ki ∈T. These candidate keyword covers are stored and maintained in the heap H. After this, the candidate with the highest score in H is selected and its function in order to generate more candidates.

$KRR*_{ki}$ – tree nodes are accessed in the depth first search order to access leaf nodes as fast as possible because the number of candidate keyword covers that are generated are very large. The first candidate solution consisting of objects other than the nodes of KRR*-tree is taken as the current best solution and it is denoted as BKC. BKC actually an intermediate solution sometimes pruning may also be applied to H when the score is less than BKC score, BKC will be continuously updated as long as all the remaining candidates are continue to process. If H contains no candidate then the algorithm terminates after returning current BKC.

Sometimes there is a chance of generating all possible keyword covers by the function Generate-candidate function. A better way is to generate keyword covers incrementally by combining individual nodes. The following diagram shows how nodes are incrementally generated by combining individual nodes in the bottom up fashion. K1, K2, K3 are three keywords and each keyword has two nodes. Highest score will be given first priority. Each time a new input node is combined in order to cover a keyword. We must observe one thing that if a combination has a score less than BKC score, then any super set of it must have a score less than BKC score. In such cases it is not necessary to generate the superset.

Algorithm 1- Baseline (T, Root)
Input:
1. A set of query keywords T and
2. The root nodes of all KRR*-trees, Root

Output:
Best scalable keyword cover
1. bkc ← ∅
2. H ← Generate-Candidatelist(T,Root,bkc)
3. While(H is not empty) do
4. Can ← the candidate in H with the highest score
5. Remove can from H
6. Depth-first-search(H,T,can,bkc)
7. For each candidate ∈ H do
8. If(candidate. score ≤ bkc.score) then
9. Remove candidate from H
10. End if
11. End while
12. Return bkc

Algorithm 2- Depth-first-search (H<T,can bkc)
Input:
1. A set of query keywords T
2. A candidate can
3. The candidate set H, and
4. The current best solution bkc

Output:
1. If(can consists of leaf nodes) then
2. s ← the set of objects in can
3. bkc' ← obtained keyword cover with the highest score in S
4. if(the value bkc.score < bkc'.score) then
5. bkc ← bkc'
6. Else
7. New-cans ← Generate-candidatelist(T,can,bkc)
8. Replace can by new-cans in H
9. can ← the candidate in new-cans with the highest score
10. Depth-first-search (H,T,can,bkc)
11. End if
12. End if

Algorithm 3: Generate –candidate –list (T,can,bkc)
Input:
1. A set of query keywords T
2. A candidate can
3. The current best solution bkc

Output:
1. New-can ← ∅
2. comvalue ← combining child nodes of can to generate keyword covers
3. foreach (comval ∈ comvalue) do
4. if(comval.score > bkc.score) then
5. New-can ← comval
6. End if
7. End for
8. Return New-can

IV. KEYWORD NEAREST NEIGHBOR DETAILS

Baseline algorithm produces the correct result for the given query, but only thing is its computational cost is very high. It references large number of objects. These objects are represented as minimum bounding rectangles. Baseline algorithm is good but its performance decreases very fastly. When the number of query keywords increases because keyword covers generated increases very fastly.

Authors have proposed a new algorithm called keyword nearest neighbor algorithm to improve the performance of baseline algorithm dramatically.

This new algorithm is mainly based on a new concept called principle query keyword. All the objects that are mapped with the principle query keyword are called principle objects. Assume that the principle query keyword is denoted by k and the set of principle objects that are associated with k are denoted by o_k.

T is a set of query keywords and the principle query keyword k∈T. The local best keyword cover of a principle object o_k., is denoted as

$$lbkc_{ok} = \left\{ \begin{matrix} kc_{ok} \mid kc_{ok} \in KC_{ok} \\ kc_{ok}.score = \max_{kc \in KC_{ok}} kc.score \end{matrix} \right\} \rightarrow (5)$$

Where kc_{ok} is the set of keyword covers in such a way that the principle object o_k is a member of each keyword cover. First for each principle object $o_k \in o_k$, $lbkc_{ok}$ must be identified. Check all the principle objects, the $lbkc_{ok}$ with highest score is called global best keyword cover (GBKC_k).

Obviously any query keyword can be selected as the principle query keyword. For each principle object local best keyword cover must be computed. In general, the query keyword with the minimum number of query keyword objects will be selected as the principle query keyword for improving the performance of the algorithm.

Computational details of local best keyword cover:

Assume that o_k be the principle object. , $lbkc_{ok}$ consists of principle object o_k and all the objects

belonging to each non-principle query keyword that is close to o_k and have highest keyword ratings. Also there is a provision to compute $lbkc_{ok}$ by incrementally retrieving the keyword nearest neighbors of o_k .

Definition of keyword Nearest Neighbor:

Assume that T is the set of query keywords and let k be the principle query keyword and K∈T and a non-principle query keyword k_i is such that $k_i \in$ {T-K}. Suppose that o_k be the set of principle objects and o_{ki} is the set of objects of keyword k_i. The keyword nearest neighbor of a principle object $o_k \in O_k$ in keyword k_i is $o_{ki} \in o_{ki}$ if and only if $\{o_k , o_{ki}\}.score \geq \{o_k , o'_{ki}\}$ score for all o'_{ki} , o_k .

The first keyword nearest neighbor of a principle object o_k corresponding to the keyword k_i is denoted as $o_k.mn^1_{ki}$ and the second keyword nearest neighbor is denoted as $o_k.mn^2_{ki}$, and also the third keyword nearest neighbor is denoted as $o_k.mn^3_{ki}$, and so on . All these keyword nearest neighbors are obtained by traversing the KRR* k_i -tree. Suppose consider a node N_{ki} in the KRR*k_i-tree.

$\{O_k , N_{ki}\}$.score = score (A, B)

Where A value is computed as A= dist(O_k , N_{ki}) and B value is computed as B=min($N_{ki}.maxrating, O_k.rating$) and we take dist(O_k , N_{ki}) is the minimum distance between O_k and N_{ki} in the normal geographical space of two dimensions (x-dimension and y-dimension) and N_{ki} .maxrating is the maximum keyword rating of N_{ki} along the keyword rating dimension.

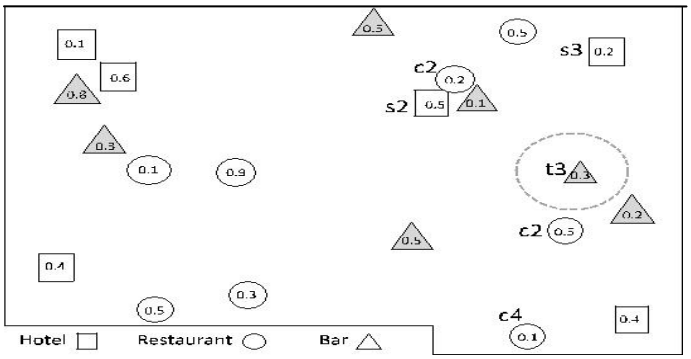
In order to compute local best keyword cover $lbkc_{ok}$ we retrieve the keyword nearest neighbor of o_k incremental way. Also note that we traverse the KRR*k_i-tree in the best first search (BFS) strategy in order to retrieve the keyword neighbors of the principle object o_k corresponding to the keyword k_i. First root node of the KRR* k_i -tree is traversed and then all of its children are stored in the heap memory H. Then for each node$N_{ki} \in H$, the score value $\{O_k , N_{ki}\}$.score is evaluated. Finally among all the nodes in H1 the node with the highest score is replaced by its child nodes and the same process is repeated until an object o_{ki} that is not in the KRR*k_i-tree is visited. Finally, O_k is selected as the best object and the score value $\{o_k , o_{ki}\}$.score is selected as current-best.

Description of keyword Nearest Neighbor Algorithm:

Keyword nearest neighbor algorithm follows a special technique for performance improvement by processing the principle keyword objects in terms of blocks rather than processing the each block separately. Assume that k is the principle query keyword. All the principle objects associated with k are indexed by using KRR*k_i-tree. Suppose that N_k is the principle node in the KRR*k_i-tree and the local best keyword cover of N_k is denoted as $lbkc_{ok}$ and it consists of N_k and the corresponding nodes of N_k each non-principle keyword of the query. Note that any KRR*k_i-tree is present at the hierarchical level1 and all the child nodes of root are present at hierarchical leve12, and all child of the nodes at level1 are presented at level2 and soon. Assume that the node N_k is presented at level6 in the KRR* k_i -tree, then the corresponding nodes of N_k in keyword k_i are all those nodes at the same hierarchical level6 in the KRR*k_i-tree. Keyword nearest neighbors is obtained incrementally in order to compute, $lbkc_{nk}$ from all the corresponding nodes.

The execution procedure detail of keyword nearest neighbor algorithm is explained below. First the algorithm selects a principle query keyword K∈ T and then starts its execution. The algorithm visits the root node of KRR*k_i-tree and then stores all the child nodes of the root in the heap H. The score value of $lbkc_{nk}.score$ is calculated for each node N_k in the heap H. The node whose maximum score is H.head is processed first. If maximum node (H.head) is present in KRR*k_i-tree, then it is replaced in H by its child nodes. First we will compute $lbkc_{nk}$ .score for each child node N_k and then H.head is updated. We will check whether H.head is a principle object o_k or not. If H.head is a principle object then $lbkc_{ok}$ is calculated and then $lbkc_{ok}$ .score is compared with the current best solution. If $lbkc_{ok}$ .score is greater than the bkc, bkc is updated to $lbkc_{ok}$. Also pruning is applied based on the result of some operations.

MCK returns(x,y,z) Bkc returns(a,b,c)



A new algorithm is proposed for improving the performance of baseline algorithm. This new algorithm is called extended version of baseline algorithm. (Extended baseline algorithm) Extended baseline algorithm is abbreviated as EBA. In the extended baseline algorithm all the root nodes of all the data structure. The special tree may be implemented either binary search tree (BST) or other efficient multiway tree data structure. We preferably implemented binary search tree method. Same procedure is used to store and manage all root nodes of all trees in multi way search tree representation also. Original baseline algorithm scans all the root nodes of KRR trees sequentially using linear search technique. But the time complexity of linear search is O (n). Linear search is costly in terms of CPU Computers. When the number of query keywords increases the time complexity of linear search increases in a linear fashion. We have used binary search tree to manage all root nodes of all KRR trees. The time complexity of binary search tree technique is O(logn). This sub linear time complexity drastically reduces the time complexity and increases t6he performance of bat baseline algorithm dramatically. Logarithmic time complexity of proposed binary search tree procedure is suitable for most of the real life applications. Binary search method solves more than 70% of the real life problems. For many other real life problems balanced multiway search tree method is the fitted tree structure to store and main all tree nodes of all KRR trees. Always time

complexity of any tree data structure is O (logn). Binary search tree is not a balanced search tree. When the binary search tree is not balanced for a particular data set, then the multiway balanced search tree is inevitable for time critical application problems.

The proposed new algorithm is modified version of the baseline algorithm. The new algorithm is given below:

Algotrithm: Extended Baseline (T, Root)
Input:

1 A set of query keywords T
2 the root nodes of all the KRR-trees, Root

Output: Scalable and optimized key word cover

1. Store all the root nodes of all the KRR-trees Bold
2. Bkc ← ∅
3. H ← Generate-Candidatelist(T,Root,bkc)
4. While(H is not empty) do
5. Can ← the candidate in H with the highest score
6. Remove can from H
7. Depth-first-search(H,T,can,bkc)
8. For each candidate ∈ H do
9. If(candidate. score ≤ bkc.score) then
10. Remove candidate from H
11. End if
12. End while
13. Return bkc

V. ANALYSIS

To help analysis, we assume a special baseline algorithm BF-baseline which is similar to the baseline algorithm but the best-first KRR*-tree browsing strategy is applied. For each query keyword, the child nodes of the KRR*-tree root are retrieved. The child nodes from different query keywords are combined to generate candidate keyword covers(kc) which are stored in a heap H. The candidate kc 2 H with the maximum score is processed by retrieving the child nodes of kc. Then, the child nodes of kc are combined to generate more candidates which replace kc in H. This process continues until a keyword cover consisting of objects only is obtained. This keyword cover is the current best solution bkc. Any candidate kc 2 H is pruned if kc:score _ bkc:score. The remaining candidates in H are processed in the same way. Once H is empty, the current bkc is returned to BKC query. In BF-baseline algorithm, only if a candidate keyword cover kc has kc:score > BKC:score, it is further processed by retrieving the child nodes of kc and combining them to generate more candidates.

However, BF-baseline algorithm is not feasible in practice. The main reason is that BF-baseline algorithm requires to maintain H in memory. The peak size of H can be very large because of the exhaustive combination until the first current best solution bkc is obtained. To release the memory bottleneck, the depth-first browsing strategy is applied in the baseline algorithm such that the current best solution is obtained as soon as possible. Compared to the best-first browsing strategy which is global optimal, the depth-first browsing strategy is a kind of greedy algorithm which is local optimal. As a consequence, if a candidate keyword cover kc has kc:score > bkc:score, kc is further processed by retrieving the child nodes of kc and combining them to generate more candidates. Note that bkc:score increases from 0 to BKC:score in the baseline algorithm. Therefore, the candidate keyword covers which are further processed in the baseline algorithm can be much more than that in BF-baseline algorithm. Given a candidate keyword cover kc, it is further processed in the same way in both the baseline algorithm and BF-baseline algorithm, i.e., retrieving the child nodes of kc and combines them to generate more candidates using Generate Candidate function in Algorithm 3. Since the candidate keyword covers further processed in the baseline algorithm can be much more than that in BF-baseline algorithm, the total candidate keyword covers generated in the baseline algorithm can be much more than that in BF-baseline algorithm.

Note that the analysis captures the key characters of the baseline algorithm in BKC query processing which are inherited from the methods for mCK query processing.

VI. CONCLUSION

Already existing MCK(m closest keywords) query works purely based on the principle that “inter object distance is computed among all the objects, then those objects whose inter object distance is minimum are selected. Authors have proposed two new algorithms to improve the performance of m closest keywords query in different way by (considering) adding extra feature called keyword rating. Baseline algorithm is good but for large sets of keywords it is not scalable and good because the intermediate key word cover sizes are very large. At the same baseline algorithm performs linear search of all the root nodes of all KRR trees. When keywords number is very large, linear scan of KRR-tree is inefficient. In this paper we propose an extension algorithm to baseline algorithm. Performance of proposed algorithm is better than the baseline algorithm.

VII. REFERENCES

1. R. Agrawal and R. Srikant, “Fast algorithms for mining association rules in large databases,” in Proc. 20th Int. Conf. Very Large Data Bases, 1994, pp. 487–499.
2. T. Brinkhoff, H. Kriegel, and B. Seeger, “Efficient processing of spatial joins using r-trees,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 1993, pp. 237–246.
3. X. Cao, G. Cong, and C. Jensen, “Retrieving top-k prestige-based relevant spatial web objects,” Proc. VLDB Endowment, vol. 3, nos.1/2, pp. 373–384, Sep. 2010.
4. X. Cao, G. Cong, C. Jensen, and B. Ooi, “Collective spatial keyword querying,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2011, pp. 373–384.
5. G. Cong, C. Jensen, and D. Wu, “Efficient retrieval of the top-k most relevant spatial web objects,” Proc. VLDB Endowment, vol. 2,no. 1, pp. 337–348, Aug. 2009.
6. R. Fagin, A. Lotem, and M. Naor, “Optimal aggregation algorithms for middleware,” J. Comput. Syst. Sci., vol. 66, pp. 614–656,2003.
7. I. D. Felipe, V. Hristidis, and N. Rishe, “Keyword search on spatial databases,” in Proc. IEEE 24th Int. Conf. Data Eng., 2008, pp. 656–665.
8. R. Hariharan, B. Hore, C. Li, and S. Mehrotra, “Processing spatialkeyword (SK) queries in geographic information retrieval (GIR) systems,” in Proc. 19th Int. Conf. Sci. Statist. Database Manage., 2007, pp. 16–23.

9. G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," ACM Trans. Database Syst., vol. 24, no. 2, pp. 256–318, 1999.
10. Z. Li, K. C. Lee, B. Zheng, W.-C. Lee, D. Lee, and X. Wang, "IRTree: An efficient index for geographic document search," IEEE Trans. Knowl. Data Eng., vol. 99, no. 4, pp. 585–599, Apr. 2010.
11. N. Mamoulis and D. Papadias, "Multiway spatial joins," ACM Trans. Database Syst., vol. 26, no. 4, pp. 424–475, 2001.
12. D. Papadias, N. Mamoulis, and B. Delis, "Algorithms for querying by spatial structure," in Proc. Int. Conf. Very Large Data Bases, 1998, pp. 546–557.
13. D. Papadias, N. Mamoulis, and Y. Theodoridis, "Processing and optimization of multiway spatial joins using r-trees," in Proc. 18th ACM SIGMOD-SIGACT-SIGART Symp. Principles Database Syst.,1999, pp. 44–55.
14. J. M. Ponte and W. B. Croft, "A language modeling approach to information retrieval," in Proc. 21st Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval, 1998, pp. 275–281.
15. J. Rocha-Junior, O. Gkorgkas, S. Jonassen, and K. Nørva g, "Efficient processing of top-k spatial keyword queries," in Proc.12th Int. Conf. Adv. Spatial Temporal Databases, 2011, pp. 205–222.
16. S. B. Roy and K. Chakrabarti, "Location-aware type ahead search on spatial databases: Semantics and efficiency," in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2011, pp. 361–372.
17. D. Zhang, B. Ooi, and A. Tung, "Locating mapped resources in web 2.0," in Proc. IEEE 26th Int. Conf. Data Eng., 2010, pp. 521–532.
18. D. Zhang, Y. Chee, A. Mondal, A. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in Proc. IEEE Int. Conf. Data Eng., 2009, pp. 688–699.