Density based Collective Spatial Keyword Query

Li Zhang[#1], Xiaoping Sun[#2] and Hai Zhuge[#3]

[#]*Knowledge Grid Lab, Key Lab of Intelligent Information Processing*
Institute of Computing Technology, Chinese Academy of Sciences, China
[1]zhangli88214@gmail.com
[2]xiaopingsun@gmail.com
[3]zhuge@ict.ac.cn

Abstract—**Geographic objects with descriptive text are gaining in prevalence in many web services such as Google map. Spatial keyword query which combines both the location information and textual description stands out in recent years. Existing works mainly focus on finding top-*k* Nearest Neighbours where each node has to match the whole querying keywords. A collective query has been proposed to retrieve a group of objects nearest to the query object such that the group's keywords cover query's keywords and has the shortest inner-object distances. But the previous method does not consider the density of data objects in the spatial space. In practice, a group of dense data objects around a query point will be more interesting than those sparse data objects. Inner distance of data objects of a group cannot reflect the density of the group. To overcome this shortage, we proposed an approximate algorithm to process the collective spatial keyword query based on density and inner distance. The empirical study shows that our algorithm can effectively retrieve the data objects in dense areas.**

I. INTRODUCTION

With the prevalence of geo-position devices GPS and Web GIS (Geographic Information System), demand on exploiting various location-based information increases rapidly on the Web. Data objects associated with location information and textual descriptions are extensively available on the Web. People can locate these spatial objects such as restaurants, hotels, stores, hospitals, parking lots through a Web query interface. Spatial keyword querying has become a hot spot for both the database research community and the Web research community. A spatial keyword query typically contains a location point and a set of keywords. The result consists of a set of data objects that satisfy certain location constraints and keyword matching conditions. Basically, there are more than one candidates and the query processing algorithms will return top-*k* items according to a certain ranking scheme. For example, a spatial range query is to find objects around the query location points with each object matching all the query keywords. A spatial cluster query is to find a set of object clustered in a certain range whose distance to the query location is minimal. In general, one can define different constraints and conditions on the query location and query keywords to represent various query semantics. Thus, spatial keyword query has a high flexibility and is attracting more and more study from both the database community and the Web community.

Most previous works apply a complete matching scheme on query keywords. That is, each data object in the result list should contain all the query keywords. In some cases, we are more interested in a group of data objects whose keywords may not exactly match all the query keywords. In [3], a collective spatial keyword query was proposed to locate a group of objects nearest to the query location with shortest inner distance among nodes in the group, and the query keywords are contained in the union of keywords of the objects of the group.

Collective spatial keyword query can be very useful in finding an area that contains a set of objects that form a relative closed community with short distance to the query location point. For example, a user may want to drive a car to have a dinner in a restaurant and go shopping nearby afterward. Thus the user has three places to go, a parking lot, a restaurant, and a shopping mall. As we present in Fig. 1, a collective query q = {s, r, p} is to find a restaurant, a store and a parking lot. A set of objects $o1$, $o2$, and $o3$ is returned by the method [3] because they have the closest distance to the query point, the minimal inner distance, and they cover all of the query keyword. So the user can visit these three locations in a small area.

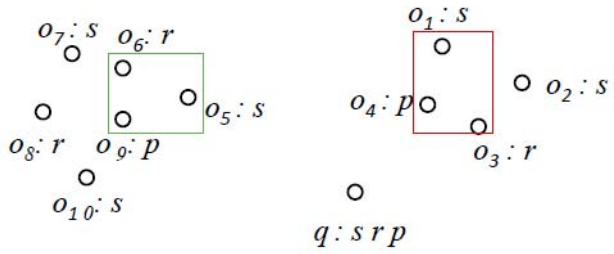


Fig. 1. A collective spatial keyword query sample.

However, a close check shows that the result returned by the method in [3] is not always a desired one as it omits the density of the area of objects. For example, in Fig. 1, o_5, o_6, and o_9 in the left block are not in the optimal solution. But there are much more candidates around them with related keywords that the result o_1, o_2, and o_3. That is, if a user visits the left area in Fig. 1, he or she can have more choices than visiting the right area. The key problem of the previous method in [3] lies in that it does not consider the density of areas of candidate objects because it can only return a group with least number of objects that cover the query keywords and match the distance constraint. It does not consider

978-0-7695-4794-7/12 $26.00 © 2012 IEEE
DOI 10.1109/SKG.2012.27

keyword distributions of the areas of candidate locations. That is, the area with denser similar objects should be given higher priority.

To improve the collective spatial keyword query, we devised a new algorithm which takes the object density related to query's keywords into account. When applying our method in the example of Fig. 1, o_5, o_6, and o_9 are returned. To the best of our knowledge there is not yet any research available that combines the object density with collective spatial keyword query.

II. Related work

Much research [7] in geographical information retrieval has been conducted to attack the problem of extracting geographic information from the Web pages. Various kinds of keyword queries on spatial Web objects were studied. In general, Web spatial objects contain both location and text description information. In most Web applications, text descriptions of objects use natural language and semi-structured data structure. Thus, keyword query is suitable for finding relevant spatial objects just like those keyword queries in information retrieval applications on the Web. In a spatial keyword query application, we input some keywords; the application provides a set of spatial web objects on map, each satisfying the query's keywords and a certain location condition.

In [11], *m*-closest keywords (mCK) query is studied, where each object has a set of keywords as its description and the query is to find a group of nodes with minimum diameter and each node has the query keyword in its description text. In [1], approximate spatial keyword query is investigated to support non-exact matching of query keyword to the data object descriptions. kNN and range spatial query are supported. In [5] and [6], top-K objects nearest to the query points are returned with the keywords of each node covering the query keywords. Ranking of keyword matching is also considered so that one can quickly locate the objects through the keyword inverted index. In [4], they tackle the problem by supporting large numbers of footprint representations. In [9], Lu et al. aim to find reversed *top-k* nearest neighbors. Cong et al. [2] studied the problem how to find top-k web objects by prestige score.

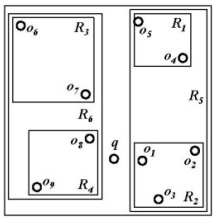
In most previous works, query keywords need to be fully covered by the descriptions of objects. In many cases, users may not need such a strict matching of keyword for each node. Instead, if the union of keywords of a group of nodes can cover the query keywords, we also can return this group of nodes as the final solution. This collective spatial keyword can be used to find an area with a group of nodes that combine different functions to form a synthetic function area. For example, in a shopping area where stores, parking lot and restaurants can be located nearly. Users can use a set of keywords to locate such area but those keywords can only be covered by a group of objects and any individual object cannot satisfy that query. In [3], an index is proposed to efficiently process the collective spatial keyword query. The method in [3] takes the distance between query object and data objects, and inner-objects distance in a group as key factors to determine the candidate objects. But it ignores that in the real world similar objects are often distributed together. In many cases, it will return a group of objects in a small cluster and ignore those areas with large number of candidate objects. To consider the keyword distribution in a candidate area, one needs to consider the density of objects. In this paper, we proposed a density based collective spatial keyword query that can determine the group of objects according to the density of keyword distribution as well as the distance to the query location.

III. Improved collective spatial keyword querying

Spatial databases mainly deal with multi-dimensional data space and R-tree is a widely used index to efficiently support range query, kNN query, etc [8]. However, the classical R-tree index only supports queries with coordinate inputs in a metric space. Keyword queries of objects cannot be directly implemented in R-tree. Thus, one key challenge in spatial keyword queries is to combine the keyword queries with spatial queries. IR-tree is a basic index structure that is used to combine the inverted index [12] with the R-tree index to support spatial keyword queries [3]. Our work is based on the IR-tree structure to combine the collective keyword query with the spatial query. First, we will introduce the IR-tree and the collective spatial keyword query index, and then introduce the density based spatial keyword query algorithm.

A. IR-Tree

The hybrid index which combines the spatial indexing e.g. R-tree with the text indexing (e.g, inverted lists) is prevalent when processing the spatial objects with text descriptions. Using R-tree, we can prune the branch after confirming there is no keyword in corresponding nodes and thus achieve an efficient query process.



object	*words*
o_1	t_1, t_2
o_2	t_2, t_3
o_3	t_1, t_3
o_4	t_1, t_4
o_5	t_4, t_5
o_6	t_1, t_4
o_7	t_4, t_5
o_8	t_2, t_5
o_9	t_3, t_5

(a)Object locations (b) object descriptions

Fig. 2. The data objects with keywords in a 2D-dimension space.

The IR-Tree [5] is a hybrid indexing integrating an R-tree with inverted index structure. In an IR-tree index, each node is constructed using traditional R-tree algorithm according to the geographic locations of data objects. Beside the pointers to the child nodes, each R-tree node also has a pointer to an inverted index file which consists of all the keywords belonging to the data objects in this node. And, in an invert index, for each keyword there is a list of node in which this keyword exists.

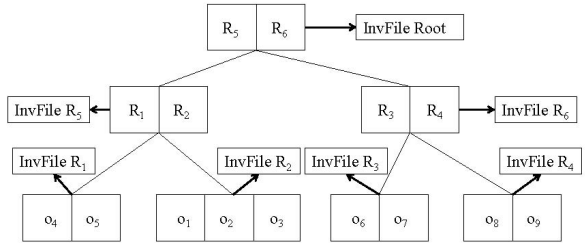


Fig. 3. An IR-tree index

For example, Fig. 2a contains eight spatial objects that are indexed by an R-tree (rectangular are the internal nodes of the R-tree index), and Fig. 2b shows the keywords appearing in the description of each object. Fig. 3 illustrates the corresponding IR-tree. Table 1 shows the content of the inverted files associated with each node. The root node contains all of the keywords of objects in its inverted file. Then, each node contains all of the keywords in its subtree. So, when there is a query sent into the IR-tree index, it was first compared with the child nodes, checking whether the keywords of child nodes match the query keyword and forward the query to the child nodes with relevant keywords and matching the distance conditions. We adopt the method from [5] to build the IR-tree index structure. Based on the IR-tree index, we implement an approximate collective spatial keyword query.

Table 1. Inverted list of nodes of an IR tree index.

Root	R_5	R_6	R_1	R_2	R_3	R_4
$t_1(4):R_5,R_6$ $t_2(3):R_5,R_6$ $t_3(3):R_5,R_6$ $t_4(4):R_5,R_6$ $t_5(4):R_5,R_6$	$t_1(3):R_1,R_2$ $t_2(2):R_2$ $t_3(2):R_2$ $t_4(2):R_1$ $t_5(1):R_1$	$t_1(1):R_3$ $t_2(1):R_4$ $t_3(1):R_4$ $t_4(2):R_3$ $t_5(3):R_3,R_4$	$t_1(1):o_4$ $t_4(2):o_4,o_5$ $t_5(1):o_5$	$t_1(2):o_1,o_3$ $t_2(2):o_1,o_2$ $t_3(2):o_2,o_3$	$t_1(1):o_6$ $t_4(2):o_6,o_7$ $t_5(1):o_7$	$t_2(1):o_8$ $t_3(1):o_9$ $t_5(2):o_8,o_9$

B. *Density based collective spatial keyword query*

Formally, given a set of spatial Web objects $D=\{o_i\}$ where $o_i=<p_i, k_i>$, p_i is the location coordinates and $k_i=\{s_j\}$ for j = 1, 2, ..., is a keyword string set as the text description of object o_i. p_i is in a 2-d Euclidean space with a distance metric d. A query $q=<\lambda, \phi>$ where λ is the query location and $\phi=\{s_m\}$ for m = 1, 2, ..., is the query keyword set.

Then, in [3], the result of a query q on a data set D is a set of objects $R=\{r_i\}$ for $r_i \in D$ for i = 1, 2,... m, satisfying the following conditions:

c1: keyword matching condition: $q.\phi \subseteq \cup p_i$ for i = 1, 2, ..., m; That is, the union of keywords of resulted objects should cover the query keyword.

c2: min($\Sigma d(\phi, r_i)$); That is, the distance from the result objects to the query location should be minimized.

c3: min($\Sigma d(r_i, r_j)$); That is, the distance between two objects in the result group should be minimized.

Obviously, to achieve above three goals is a NP-hard problem because it is even harder than finding a Steiner tree in a 2-d Euclidean space. Thus, approximate algorithm is proposed in [3] to retrieve objects from the root node to the leaf node on the IR-tree according to a cost function defined on the keyword matching score and distance score.

However, the distance condition does not reflect the density of areas of objects. We replace the third condition *c3* with the density condition:

c3': max{|R|/A_R}, where A_R is the size of area the group R occupies. That is, we want to locate the group of objects R whose area is of most density.

Locating a group satisfying condition *c3'* is also a hard problem. But it has the different semantics from the inner distance defined in *c3*. In fact, groups that satisfy *c3* does not necessarily satisfy *c3*'. For example, in Fig. 1, the right side group satisfies *c3* but not *c3'* because the left side area has a larger size and a higher density.

To process the density based collective spatial keyword query, we design an approximate algorithm based on the IR-tree to locate the results from the top node down to the leaf nodes of the index tree.

Algorithm 1 describes the major procedure of the query processing. The main loop is to extract a node from a priority queue and then for each of its child nodes, check if the keyword of the node satisfies the query keywords and put the child node into the queue according to its density cost in the current IR-tree node, until all the query keywords are covered by the founded nodes. Algorithm 1 uses a min-priority queue to record every non-leaf node such that the node with higher density and lower distance from query object will be preferred (line 11-14). When traversing the IR-tree, we prune the node whose keywords cannot completely cover the query's keywords (line 10). The function *RegionQuery* (Algorithm 2) is used to find group nodes with minimum inner-distance, which is adopted from the approximate algorithm proposed by paper [3].

Algorithm 1: Improved Alg(q,$irTree$)

Input: q = (q.λ, q.φ)
Output: a set of objects covering all the keywords and having a relatively small cost in distance and density

```
1  U ← new min-priority queue;
2  e ← irTree.root;
3  keywordsNum ← |e.φ|;
4  density ← keywordsNum/e.area();
5  priorityValue ← minDist(e, q)/density;
6  U.Enqueue(e,priorityValue);
7  while U is not empty do
8      e ← U.Dequeue();
9      if e is non-leaf node then
10         if e.φ ⊇ q.φ then
11             keywordsNum ← |e.φ|;
12             density ← keywordsNum/e.area();
13             priorityValue ← minDist(e, q)/density;
14             U.Enqueue(e,priorityValue);
15 e ← U.Dequeue();
16 V ← RegionQuery(q,e);
17 return V;
```

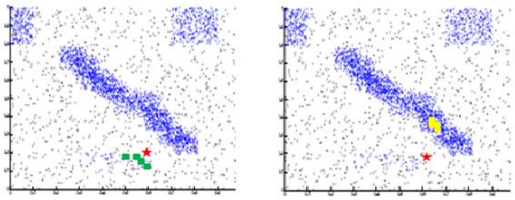
Algorithm 2: RegionQuery(q,$node$)

Input: q = (q.λ, q.φ)
Output: a set of objects covering all the keywords and having a relatively small cost

```
1  U ← new min-priority queue;
2  U.Enqueue(node,0);
3  V ← φ ;
4  uSkiSet ← q.φ;
5  while U is not empty do
6      e ← U.Dequeue();
7      if e is an object then
8          V ← V ∪ e;
9          uSkiSet ← uSkiSet\e.φ;
10         if uSkiSet = Φ then
11             break;
12     else
13         read the posting list of e for keywords in uSkiuSet;
14         foreach entry e'.φ ≠ Φ do
15             if uSkiuSet ∩ e'.φ ≠ Φ then
16                 if e is a non-leaf node then
17                     U.Enqueue(e',minDist(q,e'))
18                 else
19                     U.Enqueue(o,Dist(q,o));
20 return V;
```

In *RegionQuery* function, a priority queue is also used to visit nodes and collect the candidates. It first visits the root node and then put the child nodes to the priority queue according to their distance to the query location if they satisfy the query keywords. The priority of en-queuing object is determined by the distance to the query location (line 17 and line 19).

In summary, our algorithm first generates a set of candidate nodes using Algorithm 1 which can find a set of nodes in a dense area marked by yellow dots. Then, the result node lists are input into the Algorithm 2 as the queries to retrieve the final result with nodes having close distances to the query location.



(a) collective result (b) improved result

Fig. 4. Results of two algorithms.

IV. Experiment Result

A. Experiment Description

To show the effectiveness of our proposed algorithm, we perform two algorithms on a synthetic dataset. We compare our algorithm with [3]. Fig. 4 shows the results of two algorithms. In the Fig. 4, blue dots are nodes distributed in a 2-d Euclidean space. One can see that the many data objects are clustered in three areas. In these clustered areas, the density of objects is higher than the other area where nodes are evenly distributed. These three clustered areas belong to different function areas that have different keyword set. That is, nodes in one cluster share more common keywords than those in different areas. In this experiment, the left rectangular cluster share the common keyword set with the middle rectangular cluster. So, if a query is to find either cluster A or cluster B, then B is better if the density of cluster is considered because B is a larger cluster has higher density. The result of algorithm in [3] returns cluster A while our method return cluster B, which demonstrate that our proposed method has better performance in finding areas with higher density.

V. Conclusions

We have presented an improved collective spatial keyword query that incorporates the concept of objects' keyword density to improve the query result. The original collective spatial querying focuses on distance between query point and objects, and the inner-object distance. It may ignore surrounding objects' distribution related to the query's keywords. Instead, our improved algorithm takes object's keyword density into consideration so as to determine the nodes with higher keyword density and lower distance from query point. Thus, we can provide users more candidates implicitly. Experiments demonstrate the effectiveness of our algorithm. Ongoing work is to incorporate classification space into the proposed approach [13][14].

Acknowledgment

This work was supported by National Science Foundation of China (61070183 and 61075074).

[1] Bin Yao, Feifei Li, M. Hadjieleftheriou, Kun Hou, Approximate String Search in Spatial Databases, ICDE 2010, 2010.

[2] Cao, G. Cong, and C. Jensen. Retrieving top-k prestige-based relevant spatial web objects. VLDB2010, 3(1-2):373-384, 2010.

[3] X. Cao, G. Cong, C. Jensen, and B. Ooi. Collective spatial keyword querying. ACM SIGMOD 2011, pages 373-384. ACM, 2011.

[4] Y. Chen, T. Suel, and A. Markowetz. E_cient query processing in geographic web search engines. ACM SIGMOD 2006, pages 277-288., 2006.

[5] G. Cong, C. Jensen, and D. Wu. E_cient retrieval of the top-k most relevant spatial web objects. VLDB 2009, 2(1):337-348, 2009.

[6] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. IICDE 2008, pp 656-665.

[7] J. Ding, L. Gravano, and N. Shivakumar. Computing geographical scopes of web resources. VLDB 2000, pages 545-556

[8] A. Guttman. R-trees: A dynamic index structure for spatial searching, volume 14. ACM, 1984.

[9] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. ACM SIGMOD 2011 pages 349-360, 2011.

[10] D. Wu, M. Yiu, C. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. IEEE ICDE, pp. 541-552. 2011.

[11] D. Zhang, Y. Chee, A. Mondal, A. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In Data Engineering, IEEE ICDE 2009., pages 688-699. 2009.

[12] J. Zobel and A. Mo_at. Inverted files for text search engines. ACM Computing Surveys, 38(2):6, 2006.

[13] H.Zhuge, Y.Xing and P.Shi, Computing geographical scopes of web resources , *ACM Transactions on Internet Technology*, 8/4, 2008.

[14] H.Zhuge and Y.Xing, Probabilistic Resource Space Model for Managing Resources in Cyber-Physical Society, *IEEE Transactions on Service Computing*, July-Sept, 2012.