Future Generation Computer Systems 32 (2014) 211–221

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Density-based spatial keyword querying

Li Zhang [a], Xiaoping Sun [a], Hai Zhuge [b,c,*]

[a] Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences, China
[b] Nanjing University of Posts and Telecommunications, Nanjing, China
[c] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

HIGHLIGHTS

- A density based collective spatial keyword query is proposed.
- A cost function on IR-tree is proposed for evaluating density and distance.
- A heuristic search algorithm for finding objects in dense areas is proposed.

ARTICLE INFO

Article history:
Received 5 September 2012
Received in revised form
30 January 2013
Accepted 4 February 2013
Available online 27 February 2013

Keywords:
Density
Spatial database
Keyword query
IR-tree index

ABSTRACT

With the rocket development of the Internet, WWW(World Wide Web), mobile computing and GPS (Global Positioning System) services, location-based services like Web GIS (Geographical Information System) portals are becoming more and more popular. Spatial keyword queries over GIS spatial data receive much more attention from both academic and industry communities than ever before. In general, a spatial keyword query containing spatial location information and keywords is to locate a set of spatial objects that satisfy the location condition and keyword query semantics. Researchers have proposed many solutions to various spatial keyword queries such as top-K keyword query, reversed kNN keyword query, moving object keyword query, collective keyword query, etc. In this paper, we propose a density-based spatial keyword query which is to locate a set of spatial objects that not only satisfies the query's textual and distance condition, but also has a high density in their area. We use the collective keyword query semantics to find in a dense area, a group of spatial objects whose keywords collectively match the query keywords. To efficiently process the density based spatial keyword query, we use an IR-tree index as the base data structure to index spatial objects and their text contents and define a cost function over the IR-tree indexing nodes to approximately compute the density information of areas. We design a heuristic algorithm that can efficiently prune the region according to both the distance and region density in processing a query over the IR-tree index. Experimental results on datasets show that our method achieves desired results with high performance.

1. Introduction

Cyber Physical Society calls for a synthetic information computing framework that integrates the physical world objects with computing devices for providing ubiquitous and intelligent services [1,2]. With the quick development of GPS services, mobile devices and the World Wide Web, computing platforms integrating physical position and Web data information are becoming a major service in the Cyber Physical Society. Web GIS portals are becoming a popular information source for people to find location-based information such as place and routing information. The explosive growth of online spatial objects associated with location and textual tags drives researchers to design more efficient processing scheme for querying spatial objects in the Cyber Physical Society. Just like the text Web search engines, keyword queries on spatial data have become one major focus among spatial object query researches in database, GIS, Web, and information retrieval areas, not only because spatial keyword query can provide a flexible and extensible query interface, but also because many new research challenges emerge in handling spatial keyword queries [3]. Spatial keyword query [4–8] typically takes a location and a set of keywords as input parameters and returns the matched objects according to a certain spatial constraints and textual patterns. In these researches, each spatial object has a location coordinate in a 2D space and is tagged with a set of keywords to represent the text descriptions/content, categories, types, or attributes of the object.

* Corresponding author at: Nanjing University of Posts and Telecommunications, Nanjing, China.
E-mail addresses: zhangli@kg.ict.ac.cn (L. Zhang), sunxp@kg.ict.ac.cn (X. Sun), zhuge@ict.ac.cn (H. Zhuge).

0167-739X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.future.2013.02.007

A spatial keyword query q has a set K_q of query keywords to match keyword tags of objects and a query location coordinate L_q to limit the positions of objects. For example, a simple kNN spatial keyword query is to find objects that contain all the query keywords K_q and are the first k objects closest to the query point L_q. One can also use the query location point to find a group of objects with closest inner distance and is also closest to the query point. So, spatial keyword query has a very rich query semantics that can be used to implement various important query services on the Web GIS portal.

Different from the previous classical spatial database query, spatial keyword query involves uncertainties when locating spatial data objects. There is often no exactly correct answer for a given query. That is, for spatial data objects with textual descriptions, we cannot define an absolute correct rule to tell which one is the best for a given set of query keywords. So, in most cases, ad hoc patterns are used to find a set of reasonable objects. This problem is actually inherited from classical information retrieval area where subjective standards are often used to judge the quality of results in mining information of interests, and, indeed, spatial keyword query has become a research problem covering traditional GIS, database, and information retrieval research. Many techniques from text information retrieval have been applied in the spatial keyword query process to implement intelligent data services for spatial object finding [6,7].

In general, when combining textual data with spatial information, there are two ranks for selecting proper objects according to a query condition: one is the text rank that determines to what degree the textual descriptions of an object match the query keyword; the other is the spatial rank which ranks objects according to the distance constraints of the query. Moreover, a final rank combining these two ranking factors will be computed to rank objects in a unified way and return relevant objects according to the final ranks. People have already conducted extensive research in these two separated areas. But it is still a challenging problem when we combine these two ranking problems into one computing framework. Computing one rank after another takes too much time. Current solutions provide an integrated ranking system that can compute two ranks in one run using certain approximate ranking algorithm. So, the key problem is how to implement an efficient synthetic object filter and ranking framework.

In this paper, we study how to efficiently support a density based spatial keyword query using a hybrid index IR-tree [9] that integrates a text inverted index and an R-tree index for a unified object ranking computing. The density based spatial keyword query considers not only the text description of objects and the distance of objects to the query point, but also the density of area where objects are located in. The density of an area of objects represents the popularity of the area. The higher the density of an area, the more popular the area is, and so for the objects in that area. We use the density of area, instead of the object–object distance in the area, because the density can reflect irregularity of popular areas while distance based clusters are only in a circle area. In real world, most popular areas in a city are not strictly in a circle area but can be in any shape along the road network in an area. For example, in many cities, center shopping areas in the middle town are often scattered along a narrow long belt-shaped area, where shops are located along the street. Most of the previous work uses the distance among objects to find a cluster, which will omit those irregular popular areas when computing the ranks of objects in a cluster [9,4,6,7,10].

When considering the density of objects located in an area, we will study how to support the collective keyword query that is to find a group of objects with keywords jointly satisfying the query keywords. That is, in a popular area, each object may not fully contain the query keywords but the union of objects in this area contains the whole set of query keywords, and in this case, the objects in that area will be a candidate solution. Collective keyword query can be used to find an area where different types of objects are clustered to provide different services. For example, a person can schedule a shopping plan in an area where he/she need to do shopping, see a movie and have a lunch then. In this case, a query containing "*shopping*, *movie*, and *restaurant*" cannot be matched by any single object in that area. But three closely neighboring objects in the area, each matching only one query keyword, can be a nice solution for that query. In [9] Cong et al. use an IR-tree index to support collective spatial keyword query. The query is to find a group of objects that is closest to a query point, has the shortest object–object distance inside the group and collectively matches the query keywords. However, using object–object distance, the algorithm often returns objects in non-popular areas and miss many resulted areas with much higher density.

To address this problem, we propose a density based approximate algorithm for computing the collective spatial keyword query based on an IR-tree index [6]. Density based cluster algorithms such as CLARANS [11] and DBSCAN [12] require scanning the whole points, which can be costly when the number of data points is huge. To improve the efficiency, we modify the IR-tree such that each node records the number of keyword of objects and the smallest area of Minimum Bounding Rectangle (MBR) of objects within this node. Then, we define an approximate cost evaluation function and use an iterative scanning method to search the nodes in the IR-tree index for locating objects in dense areas, skipping those nodes that are in the low density areas. We perform experiments to show that the proposed algorithm can achieve a better result than the previous collective spatial keyword query. The main contribution of this work can be summarized as follows:

(1) Incorporate the density of area into spatial keyword query processing by defining a density–distance cost evaluation function over the IR-tree index, which can avoid heavy computing on finding the clusters with high density and still achieve a better result on finding high-density areas than the previous collective spatial keyword query algorithm.
(2) Design a heuristic algorithm that can efficiently process the query over the IR-tree index using the cost evaluation function.

The rest of paper is organized as follows. We first formulate the problem of density-based keyword querying in Section 2. We introduce the IR-tree, and present our prune technique in the modified IR-tree in Section 3. We describe the density-based keyword querying algorithm in details in Section 4. Experimental results are presented in Section 5. We review the related works in Section 6. Finally we make a conclusion in Section 7.

2. Basic model

2.1. A simple example

We first use a simple example to illustrate how the density based spatial keyword query works. In Fig. 1, there are totally ten objects $O_1, O_2, \ldots, O_{10}$, scattered in a 2D Euclidean space, and each has only one keyword as its tag, represented by an alphabet (e.g., r, s, p, representing restaurant, shop, and parking lot respectively). A query q is issued with three query keywords s, r, and p. In Fig. 1, three objects o_1, o_3, and o_4 in the right box are the result of the algorithm in [9]. Obviously their tags can cover the query keyword set. They are close enough to each other and they are also close to the query point q. That is, the maximum distance among o_1, o_3, and o_4, plus their distance to q, is minimal among all the candidate objects. Note that, there is also a group of objects

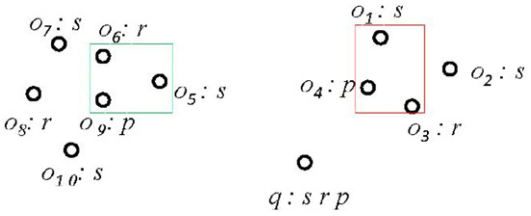


Fig. 1. A simple collective query.

o_5, o_6, and o_9, in the left box that can cover the query keywords. Moreover, these three objects are located in a denser area than the previous three objects, which means that users will have more choices in this area, although the distance to the query point is a little bit longer than that of o_1, o_3, and o_4. So, when we consider the density of area, the left part of the Fig. 1 can be a better choice than the right part. Of course, there is no absolute correct result in this application. Users may prefer a closer distance to a dense hotspot area, or consider the density when the distance cost is almost the same. So the density of area is also a necessary factor when processing the spatial keyword query.

2.2. Data and query model

To simplify the discussion, we assume, a set of n objects are located in a 2D Euclidean space, each having a coordinate as its location and being tagged with a set of string keywords. Thus, we define the data set and query as follows:

Data: Let $D = \{o_1, o_2, o_3, \ldots, o_n\}$ be a set of spatial objects. Each object $o_i = (\delta_i, \tau_i)$ has a location $\delta_i = (x_i, y_i)$ and is tagged with a keyword string set $\tau_i = \{t_1, t_2, \ldots, t_j\}$. The distance between two objects o_i and o_j is defined as a 2D Euclidean distance dist $(o_i, o_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$.

Query: A query $q = (\lambda, \varphi)$ contains a location $\lambda = (q.x, q.y)$ and a set of keywords $\varphi = \{k_1, k_2, \ldots, k_m\}$. The informal semantics of the query is to locate the minimal set of objects $R = \{o_1, o_2, \ldots, o_t\}$ s.t. $\varphi \subseteq \cup o_i \cdot \tau_i$ for $o_i \in R$, and R is located in the densest area that is also close to the query point. The density of a set R of objects in a region B is defined as the $\delta(R, B) = |R|/|B|$, where $|R|$ is the number of objects and $|B|$ is the area of the region B. Note that, B can be any shape containing R. For simplicity, we use the Minimum Bounding Rectangle of R, denoted as MBR(R), to represent the region B. Then, the distance from a query point to R is defined as dist $(q, R) = \min\{\text{dist}(q, o) | o \in \text{MBR}(R)\}$. That is, the distance to a region R is the shortest distance from an object in R to the query point.

Then, we define a cost function of a given set R of objects to a query q in Eq. (1). *FullCost* is used to rank objects for a given query. It contains two parts, one for the density and the other for the distance. α is a weight parameter controlling importance assigned to the density factor or to the distance factor when considering a group of objects. Using *FullCost* one can find a group of objects that has the minimal cost by traversing all the possible candidate objects that has keyword sets covering the query keywords. However, finding such a minimal group of objects with the minimal *FullCost* is obviously an NP-hard problem. So we need to design an efficient algorithm that can achieve an acceptable effectiveness and running performance.

$$\textit{FullCost}\ (q, R) = \alpha\delta(R, \text{MBR}(R)) + (1 - \alpha)\text{dist}(q, R). \quad (1)$$

3. Density based IR-tree index

3.1. Basic index structure

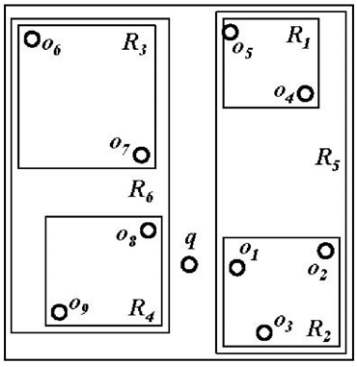
To efficiently process the density based spatial keyword query, we use an index structure IR-tree to help prune those unnecessary accesses to those objects that are obviously not in the candidate result set. The IR-tree index is a hybrid index combining R-tree and text index. Traditionally, the R-tree index [13,14] has been widely used in various spatial query such as nearest neighbor query [15,16], distance based query [17], top-K query [18], and continuous nearest neighbor query [19]. The basic idea of the R-tree index is to partition a data set using MBR. It splits a MBR R_i into two new smaller sub MBR R_k and R_l whenever the number of data points in R_i exceeds the predefined upper bound. Then, the two sub MBRs R_k and R_l are linked to the previous MBR R_i, which forms a tree index structure just like the B-tree index. The R-tree index is very efficient in processing spatial queries like the nearest neighbor query, range query, top-K query, etc., when the query is only related to the position information of objects. However, when a query involves keyword attributes of objects, we need to design new index structures to efficiently handle keyword process. Inverted index is a popular text index for processing keyword queries over text documents [20]. For each keyword, it records a list of documents that contains the keyword as well as other properties such as the frequency of the keyword in the document. One can easily retrieve the document that contains a certain keyword using the inverted index. When processing spatial keyword query, one can first use R-tree index to find a set of objects satisfying the spatial query condition, and then use the inverted index to filter the result according to the query keyword set. But this is time consuming because many unnecessary accesses occur. Current solutions use a hybrid index to process the spatial query and keyword query in one index [9,4,6,7,21]. The IR-tree index is a typical indexing structure that combines R-tree index with an inverted index to support the spatial keyword query [6].

The basic underlying structure of an IR-tree is still an R-tree index. In each node of an IR-tree index, an inverted index is attached to record the keywords of objects covered by the node. A node R covering a set of objects in an IR-tree is in the form of (R, mbr, lk, m, lc, if) where R is the identifier of the node, mbr refers to the Minimum Bounding Rectangle of the objects covered by R, lk is a list of all the keywords of objects appearing in this node, m indicates the minimum area of all the nodes in the MBR, lc denotes a child list which consists of a set of nodes in the next level and if is a pointer to an inverted file associated with the node. Each inverted file has three main parts:

- All the distinct keywords appearing in the textual tags of each object contained in the node.
- Frequency of each keyword appearing in the objects in this node.
- Each keyword t has a list of pointers to the nodes whose keyword list also contains t in the next level in the IR-tree.

Note that, the inverted index of an inner node in the index does not record the documents containing a keyword, but points to child nodes that contain a certain keyword in the IR-tree. The inverted index in a leaf node will point to the spatial object with the text content. The following is a simple example for illustrating how an IR-tree index works. Fig. 2(a) displays nine spatial objects $o_1, o_2, \ldots, o_9$ and the Fig. 2(b) shows the keywords of each object. Fig. 3 illustrates the corresponding IR-trees, and the inverted file of each node is listed in Fig. 4. For example, from Fig. 4 one can see that a keyword t_1 occurs 4 times at R_5 and R_6 and from R_5 one can further determine that t_1 is found in R_1 and R_2, and finally object o_4 can be retrieved from R_1 for t_1. Thus, for a spatial keyword query, one can scan the IR-tree index from the root node to the leaf node for locating candidate objects according to both spatial constraints and query keywords. Using a heuristic method, we can skip many objects during a traversal, which can greatly speed up the performance.

The number of entries in a node of an R-tree index can affect the overall performance of the index. Let M denote the maximum



(a) Object locations.

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

(b) Object descriptions.

Fig. 2. A simple dataset.

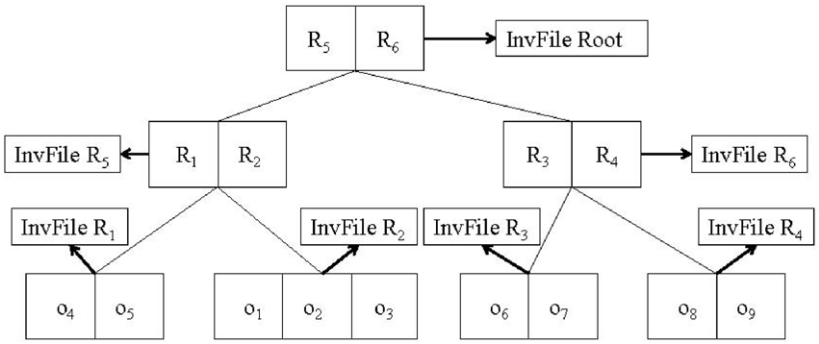


Fig. 3. An example of the IR-tree index.

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

Fig. 4. Inverted files of the nodes in the IR-tree index.

number of entries that a node can contain, and let m denote the minimum number of entries respectively. According to [13], when m is 40% of M, it yields the best performance. So in this paper, we set m to 40% of M while building an IR-tree index.

3.2. Density cost function in the IR-tree index

As computing a full cost using Eq. (1) is a hard problem, we need an approximate method to compute density and distance at the same time while filtering objects using query keywords. The IR-tree index provides such a structure that we can perform these tasks at the same time. Distance information can be directly computed in an IR-tree index. Keyword query can be processed using the inverted file attached to each node. The key is how to estimate the density information.

To incorporate the density information into an IR-tree index, we use the node size to estimate the density of areas. Let R denote the node in IR-tree. The number of keyword k contained in node R is denoted as n_k. The area of the minimum bound rectangle (MBR) of node R is denoted as $R \cdot area$, which is computed by multiplying the length and the width of the MBR in 2D Euclidean space. And let $R \cdot minArea$ denote the minimum area of nodes whose MBR is covered by that of R. We define the distance between the query and node R as dist (q, R), which is the nearest distance from the query q to the node R's Minimum Bounding Rectangle. The node density of keyword k in node R can be calculated by $d_{n_k} = n_k/R \cdot area$. Then, the node density of keyword set φ in node N can by calculated by $d_{n_\varphi} = n_\varphi/R \cdot area$, where $n_\varphi = \sum_{k \in \varphi} n_k$. That is, we use the number of keywords of objects covered by an indexing node and the area of the node to estimate the density of a region. To incorporate both distance and density information when processing a query, we define a cost function for the traversals of indexing nodes on an IR-tree index as follows.

$$Cost(q, R) = (1 + \text{dist}(q, R))^{(1-\alpha)}/d^{\alpha}_{n_\varphi}. \tag{2}$$

Eq. (2) shows that the cost of exploring an indexing node R for a query q decreases when dist (q, R) decreases and if the node density increases, the cost will also become small. α is the weight factor to control the balance between the density and the distance when traversing an indexing node. If the user set $\alpha = 1$, then the most dense node will be returned. If $\alpha = 0$, it will ignore the node density. Now we can use a query $q = (\lambda, \varphi, \alpha)$ to traverse the IR-tree index. Note that, *Cost* is computed based on the index node, instead of the real objects. So we also need a cost function to evaluate the real objects inside an index node. Cong et al. [9] define the collective cost for objects as follows:

$$CollectiveCost(q, \chi) = \beta \max_{\gamma \in \chi}(\text{dist}(\gamma, q)) + (1 - \beta) \max_{\gamma_1, \gamma_2 \in \chi}(\text{dist}(\gamma_1, \gamma_2)), \tag{3}$$

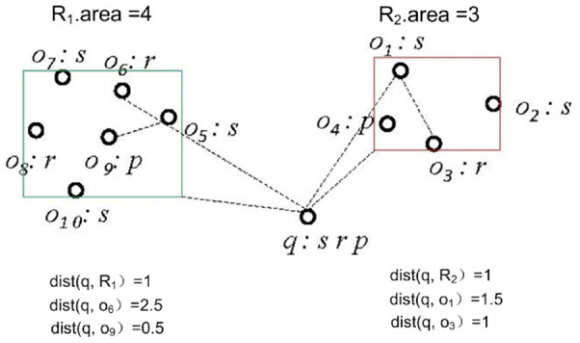


Fig. 5. Query processing example.

where χ denotes any object set and β is for weighting factor. As we can see, the collective cost consists of two parts. The former part is the maximal query-object distance and the latter part stands for the maximal object–object distance, which is to find the closely clustered objects. Thus, the density based spatial keyword query on an IR-tree index can be defined as follows:

Definition 1 (*Density-Based Spatial Keyword Querying on an IR-tree Index*)**.** Given a dataset D with an IR-tree index on it, a query $q = (\lambda, \varphi, \alpha)$ is to find an object set V, satisfying that, $V \cdot \tau \supseteq q \cdot \varphi$, $CollectiveCost(q, V)$ is minimum and V is in an indexing node R which makes $Cost(q, R)$ minimum.

Now considering the example in Section 1, we first apply the collective spatial keyword querying method from [9]. For simplicity, we take $\beta = 0.5$. The result set is $\chi = \{o_1, o_3, o_4\}$, which covers the keyword set $\{s, r, p\}$ with the lowest collective cost. Eq. (4) is the *CollectiveCost* of the result set, where $\text{dist}(q, o_1) = 1.5$ is the maximum among the right rectangle and $\text{dist}(o_1, o_3) = 1$ is the maximum object–object distance in the right rectangle. So, the cost is 1.25 for the right rectangle area while the cost for the left rectangle is 1.5 and the right area is chosen as the final result.

$$\begin{aligned} &CollectiveCost(q, \chi) \\ &= \beta \max_{\gamma \in \chi}(\text{dist}(\gamma, q)) + (1-\beta) \max_{\gamma_1, \gamma_2 \in \chi}(\text{dist}(\gamma_1, \gamma_2)) \\ &= 0.5\text{dist}(o_1, q) + 0.5\text{dist}(o_1, o_3) \\ &= 0.5 \times (1.5 + 1) = 1.25. \end{aligned} \tag{4}$$

Using the density based query cost function, we will locate the left region in Fig. 5. First we scan the IR-tree index using Eq. (2) with $\alpha = 0.5$, where we have the first two indexing nodes R_1 and R_2 and compute the areas of R_1 and R_2. The density of keyword set $\varphi = \{s, r, p\}$ in node $R_1 (resp \cdot R_2)$ can be calculated by $d_{n_{1\varphi}} = n_{1\varphi}/R_1 \cdot area = 6/4 (resp. d_{n_{2\varphi}} = n_{2\varphi}/R_2 \cdot area = 4/3)$. Then we have $Cost(q, R_1) = (1 + \text{dist}(q, R_1))^{(1-\alpha)}/d^{\alpha}_{n_{1\varphi}} = (4/3)^{0.5}$ and $Cost(q, R_2) = (1+\text{dist}(q, R_2))^{(1-\alpha)}/d^{\alpha}_{n_{2\varphi}} = (3/2)^{0.5} > Cost(q, R_1)$. Therefore, we choose R_1 for expanding by using the collective spatial keyword querying. Finally, we get the density result $V = \{o_5, o_6, o_9\}$ with the minimum collective cost in node N_1. And $V's$ tags exactly cover the query q's keywords. Note that, $CollectiveCost(q, V) = 0.5(2.5+0.5) = 1.5 > CollectiveCost(q, \chi)$. However, the result set V can give more nearby candidates to users. So users may give preference to the density based result.

3.3. Node pruning strategy in scanning IR-tree index

In this section, we devise a strategy for pruning nodes according to the keyword covering and cost minimization when scanning an IR-tree index for a query. We can utilize the IR-tree to prune the nodes that cannot cover all the query's keywords. We also observe that, if we can calculate the lower bound of the density cost of a node, then we can prune the node with the lower bound of density cost larger than the minimum density cost we have recorded so far. Let *minCost* denote the minimum density cost of the node ever searched and $LBCost(q, R)$ be the lower bound of density cost within node R. That is, if $LBCost(q, R) > minCost$, then $Cost(q, R_i) > minCost$, where R_i refers to any node encompassed in the node R such that we can prune the node R. We will prove it in the following part.

Lemma 1. *In an IR-tree, we have $n_k \geq n'_k$ and $n_\varphi \geq n'_\varphi$, where $n_k (resp.\ n'_k)$ is the number of the keyword k in the indexing node $R (resp. R')$, φ is a keyword set and R' is in the sub-tree of R.*

Proof. Obviously, if an object is in R', it must be in the node R. So we deduce that, for any keyword contained in R', it is also contained in R. So is the keyword set, since $n_\varphi = \sum_{k \in \varphi} n_k$. □

We define the upper bound of the density for a given keyword set φ in a node R as $\text{UBd}_{n_\varphi} = n_\varphi / R_{\min} \cdot area$, where $R_{\min} \cdot area$ is the minimum region area rooted in node R. Note that, for each node we record the minimal area in the IR-tree index.

Lemma 2. *In an IR-tree, we have $\text{UBd}_{n_k} \geq d_{n'_k}$ and $\text{UBd}_{n_\varphi} \geq d_{n_\varphi}$, where $\text{UBd}_{n_k} (resp. \text{UBd}_{n_\varphi})$ is the upper bound node density of keyword k (resp. a keyword set φ) in node R, $d_{n'_{k_i}} (resp.\ d_{n_\varphi})$ is the node density of keyword k (resp. a keyword set φ) in node R' which is rooted in R.*

Proof. From the definition we can find that $R_{\min} \cdot area \leq R \cdot area$, since $R_{\min} \cdot area$ is the minimum area of all the regions encompassed in node R. From Lemma 1 we have $n_{k_i} \geq n'_{k_i}$ and $n_\varphi \geq n'_\varphi$ for node R and R' in node R''. So we have:

$$\text{UBd}_{n_k} = n_k / R_{\min} \cdot area \geq n'_k / R_{\min} \cdot area \geq n'_k / R \cdot area$$

and

$$\text{UBd}_{n_\varphi} = n_\varphi / R_{\min} \cdot area \geq n'_\varphi / R_{\min} \cdot area \geq n'_\varphi / R \cdot area. \quad \square$$

Lemma 3. *In an IR-tree, we have $\text{dist}(q, R) \leq \text{dist}(q, R')$, where $\text{dist}(q, R)$ $(resp.\ \text{dist}(q, R'))$ is the shortest distance from query q to the $R's (resp.\ R')$ MBR and R' is in the sub region of R.*

Proof. If the query q is encompassed by the indexing node R, $\text{dist}(q, R) = 0$, and $\text{dist}(q, R') \geq 0$, so obviously $\text{dist}(q, R) \leq \text{dist}(q, R')$. If the query q is outside the node R, since R' is inside R, then we have $\text{dist}(q, R) \leq \text{dist}(q, R')$. □

According to Eq. (2), we can compute the lower bound density cost of node R by $LBCost(q, R) = (1 + \text{dist}(q, R))^{(1-\alpha)}/\text{UBd}^{\alpha}_{n_\varphi}$ for a query q.

Lemma 4. *Given an IR-tree, if $LBCost(q, R) \geq minCost$, we have $Cost(q, R') \geq minCost$ for any node R' contained in R, where minCost is the minimum cost of the nodes we have searched.*

Proof. From Lemmas 2 and 3, we have the following equation:

$$\begin{aligned} Cost(q, R') &= (1 + \text{dist}(q, R'))^{(1-\alpha)}/d^{\alpha}_{n'_\varphi} \\ &\geq (1 + \text{dist}(q, R))^{(1-\alpha)}/\text{UBd}^{\alpha}_{n_\varphi} \\ &= Cost\ (q, R) \geq minCost, \end{aligned}$$

which leads to the Lemma 4. □

Now we can utilize Lemma 4 to prune the unnecessary regions when scanning an IR-tree index from top to leaf. When we begin to search a node R, first we compute the $LBCost(q, R)$. If $LBCost(q, R) \geq minCost$, then we can prune the node R at once, without investigating deep into the node R.

Algorithm 1: Density-Based Search(q,$irTree$, α)

Input: q = (q.λ, q.φ, α)

Output: a set of approximate collective objects covering all the keywords with lowest density cost

```
1  U ← new a queue;
2  e ← irTree.root;
3  minCost ← ∞;
4  minCostNode ← ϕ;
5  if e.φ ⊇ q.φ then
6      U.Enqueue(e);
7  else
8      return NULL;
9  while U is not empty do
10     e ← U.Dequeue();
11     if e.subNodes are nodes not objects and e.subNodes.φ ⊇ q.φ then
12         U.Enqueue(e.subNodes);
13     n_{e.φ} ← |e.φ|;
14     UBd_{e.φ} ← n_{e.φ}/mine.area;
15     LBCost(q, e) ← (1 + dist(e, q))^{(1−α)}/(UBd_{e.φ})^α;
16     if LBCost(q, e) < minCost then
17         d_{e.φ} ← n_{e.φ}/e.area;
18         Cost(q, e) ← (1 + dist(e, q))^{(1−α)}/(d_{e.φ})^α;
19         if Cost(q, e) < minCost then
20             minCost ← Cost(q, e);
21             minCostNode ← e;
22 if minCostNode is ϕ then
23     return NULL;
24 V ← CollectiveSearch(q,minCostNode);
25 return V;
```

Fig. 6. Density-based search algorithm.

4. Density-based search algorithm

In this section, we will introduce the algorithm that utilizes the pruning techniques to process the density based spatial keyword query in an IR-tree index. Fig. 6 shows the main algorithm of query processing and an approximate collective keyword search algorithm taken from [9] is shown in Fig. 7.

The density-based search algorithm takes a query q as an input, including q's location, a set of keywords, and a density weight α which users use to adjust the balance between object-around density and query-objects distance. In the algorithm, we utilize a queue to do breadth-first search on the IR-tree. Firstly, we initialize a queue U, and set e, *minCost*, and *minCostNode* for initialization (line 1–4), where e represents the current indexing node for processing. First the root node is processed. If the root covers the keyword set of q, it will be put into the queue. Otherwise, the algorithm stops and returns null (line 5–8). In the following loop (line 9–21), each time a node is extracted from the queue and we judge if it has child nodes that can cover all the keywords. If so, we put those child nodes into the queue. Otherwise, we skip them and proceed for the current node. Line 11–12 is for the keyword comparison on child nodes. After that, we utilize the density based pruning technique to further filter the nodes (line 13–15). If $LBCost(q, e) \geq minCost$, we prune the node e. Otherwise, we calculate density cost of the current node and compare it with the *minCost*. If $Cost(q, e) < minCost$, we refresh *minCost* by $Cost(q, e)$ (line 16–21). We repeat this process until the queue is empty and get a set of candidate nodes *minCostNode* with the min cost. Finally we utilize the approximate collective search algorithm from [9] to expand *minCostNode* to get the final result (line 24).

In [9], the original approximate collective search algorithm takes the root node of an IR-tree index as the starting input node. It uses a priority queue to filter the nodes and locate the candidate node with small collective cost (see Eq. (3)). That is to say, the nearest nodes which have intersection with the remaining keyword set will be expanded first. Fig. 7 displays the approximate collective search algorithm proposed by Cong et al. [9]. They have proved that, its approximate factor is 3. In other words, the collective cost of the solution V, returned by approximate collective search algorithm, is at most three times the cost of the optimal collective solution.

5. Experimental study

There is few related work for handling density based spatial keyword query. So, we use the density based search algorithm without pruning strategy as the base line method for comparison. DBS represents the density based search algorithm while DBSNP is for the algorithm without using the pruning method proposed in Section 3. We also compared DBS with the approximate collective search algorithm (CS in CS for abbreviation) in [9]. We implemented all these algorithms in C#. All these experiments

Algorithm 2: CollectiveSearch(q,$node$)

```
Input: q = (q.λ, q.φ)
Output: a set of approximate collective objects
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;
```

Fig. 7. Approximate collective search algorithm in [9].

were run on a Windows XP machine with Intel Core Pentium(R) D 3.40 GHz CPU and 2.0 GB memory.

5.1. Effects of node size on search performance

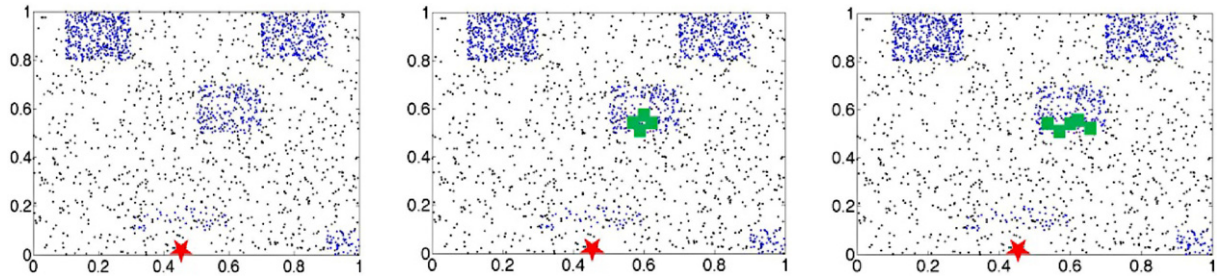
Since the density computation is based on the indexing nodes of an IR-tree, the size of indexing nodes can affect the final result. So, we first test the effects of different indexing node size on the searching result. Then, we will evaluate the effects of the weight factor α on the search results for both DBS and CS algorithm. For evaluation, we generate a set of synthetic objects in a 2D Euclidean space. For each object, we assign a set of keyword tags. If two objects share many keyword tags, they tend to be clustered in an area. In Figs. 8–10, the blue points are the spatial data objects scattered in the 2D Euclidean space and green block points are the resulted points. A query is issued at the red star point and is to locate those objects in high density areas. Fig. 8(a) shows the generated data, where we have three main clustered areas in rectangle shape located in the upper areas, and there are also lots of random tagged objects scattered in the whole space as noise points.

We use *NodeMax* to denote the size of the indexing node of IR-tree index, i.e., the maximum number of points that can be located in one indexing node. As [13] pointed out that different indexing nodes size can affect the search performance of an R-tree, we set minimum node size as 40% of *NodeMax*, which can help achieve a better search performance in a classical R-tree index. We test *NodeMax* $=$ 50, 100, 150, 200, 250 with $\alpha = 0.3$. Fig. 8(b) and (c) shows the results using the DBS algorithm with *NodeMax* $= 50$ and 250 respectively. We get the similar results for *NodeMax* $=$ 100, 150 and 200, so we do not depict them in Fig. 8. We can see that in all the cases, we can find objects in the closest clustered areas. The distribution of result is of a little difference when using different *NodeMax*. When using smaller value of *NodeMax*, the returned objects are more closely clustered than that using larger *NodeMax*. It is mainly because that, the smaller node will be more sensitive to the object distribution and thus reflects the density of areas more accurately. Fig. 9 shows results of the CS algorithm with *NodeMax* $= 50$, 250 and $\alpha = 0.3$. Obviously, the CS algorithm does not consider the density of areas and the result contains only those objects (in green block points in figures) close to the query location and the node size does not affect the final result either.

5.2. Effects of weight factor on search result

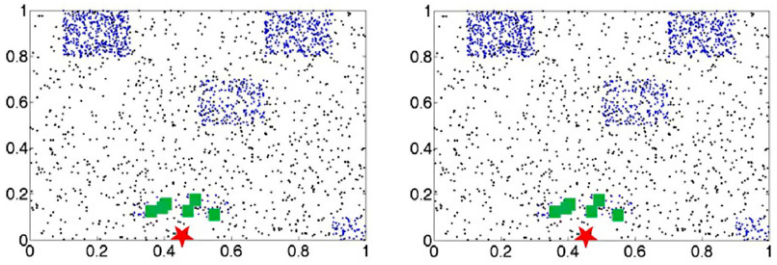
We also evaluate the results with different weight factor α. We set $\alpha = 0.1, 0.3$ and 0.9 for the DBS algorithm to run over the data set in Fig. 8(a). α is the parameter used to adjust the balance between the density of an area and the distance to the query location when searching in an IR-tree index. The smaller the parameter, the more importance the distance to query location has. For instance, as Fig. 10(a) depicts, when $\alpha = 0.1$, the DBS result is quite near the query point. In other words, the objects-query distance plays more important role than the area density. In contrast, when $\alpha = 0.9$, the DBS algorithm returns a region with higher node density but the distance to the query location is almost omitted. In Fig. 10(c), objects in the cluster in the left-up corner are located, which is almost the farthest region from the query point (red star). So, one can see that, the DBS can provide a flexible parameter allowing users to choose different preferential in querying.

We also conduct experiments on real road network map of the city of San Francisco with irregular distribution of the spatial objects (in red points in Fig. 11). Fig. 11 shows the results returned



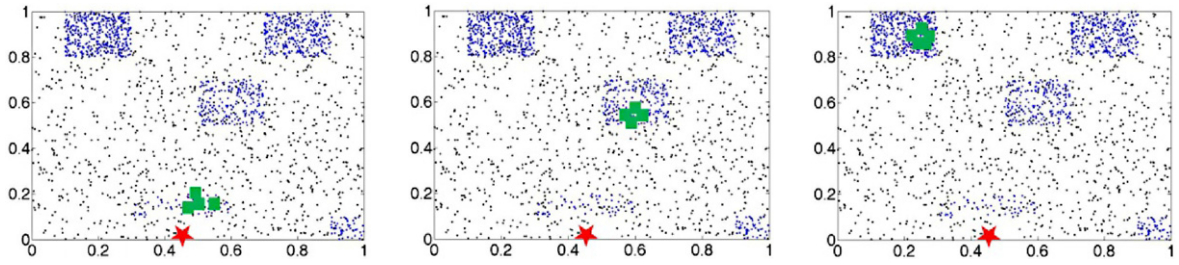
(a) Object distribution. (b) *NodeMax* = 50. (c) *NodeMax* = 250.

Fig. 8. DBS results with different *NodeMax*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



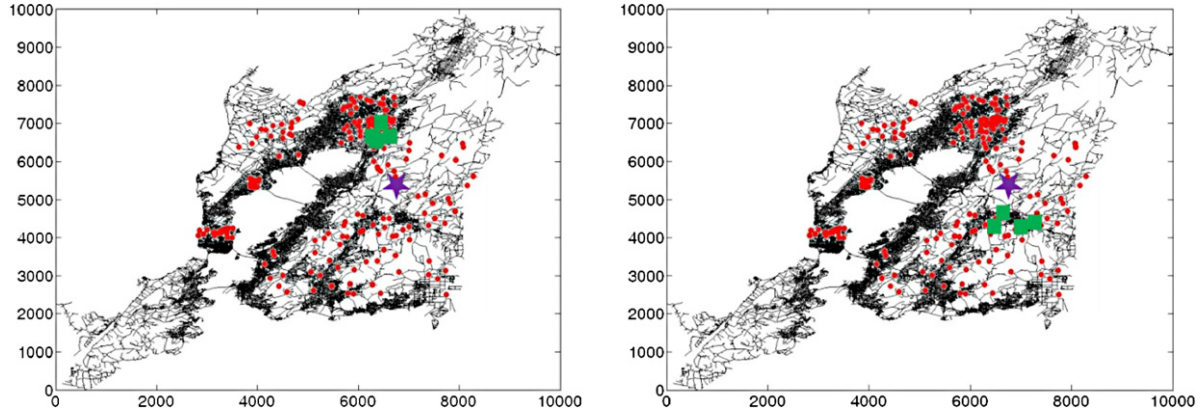
(a) *NodeMax* = 50. (b) *NodeMax* = 250.

Fig. 9. CS results with different *NodeMax*. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) $\alpha = 0.1$. (b) $\alpha = 0.3$. (c) $\alpha = 0.9$.

Fig. 10. DBS results with different weight factor α. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)



(a) *DBS* result. (b) *CS* result.

Fig. 11. DBS and CS results on a real road network map of San Francisco.

by CS and DBS ($\alpha = 0.3$, *NodeMax* = 50) algorithms. The purple star on the map is the query point randomly chosen by algorithm, and the green rectangles show the results. We perform random queries for 100 times, and the results show that, the objects in dense area are returned (in green rectangular points), while using CS algorithm, the objects near the query location are found.

5.3. Evaluation on the pruning technique

In this section, we evaluate the performance of the pruning technique used in the query processing. We implemented two algorithms for comparison: DBS is the algorithm with the prune techniques shown in Fig. 6; DBSNP is the revised DBS algorithm

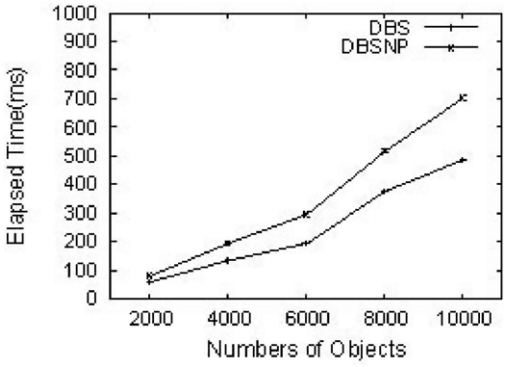


Fig. 12. Performance for different numbers of objects.

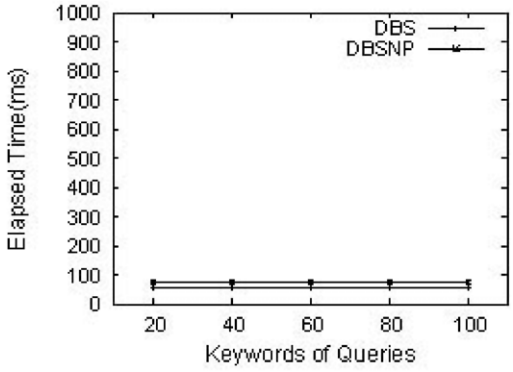


Fig. 13. Performance for different number of query keywords.

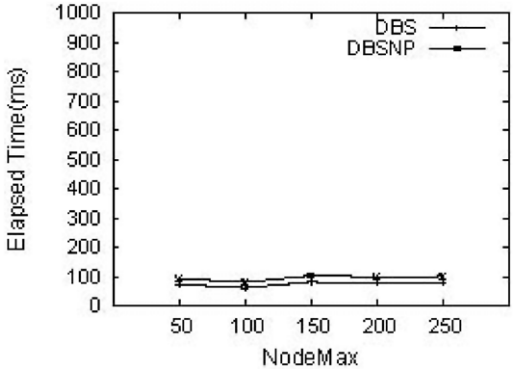


Fig. 14. Performance with different *NodeMax value*.

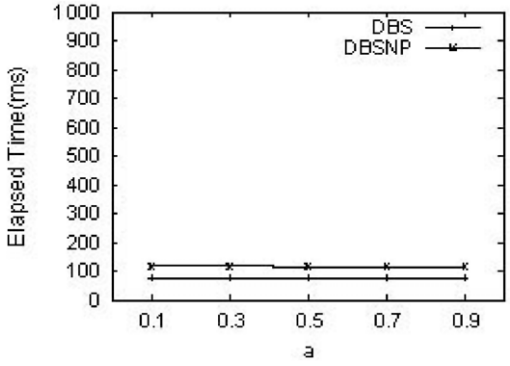


Fig. 15. Varying the weighting factor α.

by removing prune steps in DBS. That is, we skip these codes that perform the density cost based comparison with the minimum cost. Instead, we just visit every node to find the nodes with the minimum cost and then return it for further processing. We consider the following four factors that can affect the performance of the algorithm.

5.3.1. The number of objects

We first evaluate the two algorithms in different maps with 2000, 4000, 6000, 8000, and 10 000 objects respectively. We test 500 queries randomly in each map with *NodeMax* = 100 and α = 0.3. Fig. 12 shows the results. We can see that, DBS significantly outperformed DBSNP. This is because DBS not only uses keyword related pruning technique, but also can prune many unnecessary regions before searching deep into them. The processing time is linear in terms of the number of objects for both algorithms. As the number of objects increases, DBS shows more potential in reducing the searching regions. For example, in Fig. 12, when the number of objects is 2000, the average elapsed time by DBS and DBSPN is nearly the same. But when the number of objects comes to 10 000, the searching time of DBSPN almost reaches two times that of DBS.

5.3.2. The number of query keywords

Secondly, we evaluate DBS and DBSNP with different numbers of query keywords in the same map. For each type of query we randomly generate 500 different queries over 2000 objects with *NodeMax* = 100 and α = 0.3. Then we compute the average elapsed time. Fig. 13 displays the result yielded by DBS and DBSNP. We can see that, DBS outperforms DBSNP. Also note that, the number of keywords has no obvious influence on the query performance.

5.3.3. The size of indexing node

Next we evaluate two algorithms using different *NodeMax* value = 50, 100, 150, 200 and250 respectively. In the same way we put 500 random queries on one map with the number of query keywords being 20 and α = 0.3. Fig. 14 shows that, again the size of indexing node does not affect the processing time of two algorithms. Note that, the size of indexing node can affect the result set, which has been shown in the previous section.

5.3.4. The weighting factor α

In the end, we evaluate the pruning techniques by varying α in the map with 2000 objects. And we respectively set α = 0.1, 0.3, 0.5, 0.7 and 0.9 and for each α we search 500 times on the map in different positions with the same keyword set and the *NodeMax* is set to 100. We present the results in Fig. 15. From the picture we can find that, DBS(resp. DBSNP) nearly takes the same time to finish searching when we set α in different values. And varying α cannot enlarge or narrow the gap between DBS and DBSPN. That is to say α does not affect the query processing time.

6. Related works

There are already large numbers of research studies on keyword query over spatial objects with textual contents in natural language or semi-structured data structure. Keyword query is a proper query tool for users to grasp and for backend database to process. In a spatial keyword query, a set of query keywords is used to express the matching condition on textual contents of objects and a query location is to define the spatial distance condition that the candidate objects need to follow.

In [4], an IR^2-tree index is proposed to efficiently handle the nearest neighbor spatial keyword query. In [5], an index and several search strategies are proposed to efficiently process the spatial containment and intersection query with keyword. In [6,7], top-*K* query with keyword on spatial objects are studied to find top-*K* objects nearest to the query points are returned with the keywords of each node covering the query keywords. Li et al. added

direction constraint on spatial keyword querying [8]. Patroumpas et al. studied how to efficiently obtain the orientations of moving objects [22]. Zhang et al. studied how to find the closest objects that match the keywords [23,24]. Leung et al. considered location on personalized Web search [25]. In [26] a top-k query over spatial objects with non-space scores is studied, where non-space score can be any ranking factor for spatial objects. In [27], an approximate spatial keyword query is proposed to support fuzzy matching of query keyword string on the data object descriptions and kNN and range spatial query are supported using a hybrid R-tree index with textual index. In [10], the authors studied a reversed top-k keyword query. In [28], the authors studied the m-closest keywords (mCK) query, that is to find a group of nodes with minimum diameter and each node has the query keyword in its description text. Zhou et al. exploited hybrid index structures for location based web search [29]. A KR*-tree index is designed to combine an inverted keyword index and an R-tree index for processing spatial keyword query with AND operators [30]. Obviously all these studies are unable to cope with our problem.

In previous works, a result object needs to cover the whole set of the query keywords. But in real applications, it is often too strict to find such objects matching each query keyword. Collective spatial keyword query is used to locate a set of objects with the union of their keywords covering the whole query keyword set [9]. In [9], an R-tree index with inverted file index is used to efficiently process the collective spatial keyword query. The main idea is to use a well-defined cost function to heuristically search the index tree, which can achieve a good result set with nice performance. But it still does not consider the density of an area where the objects are located in. Our density-based spatial keyword querying is different from their methods in that we consider the density constraint when searching candidate objects. Our method provides a synthetic cost function to make a compromise between the query-objects distance and the object-around density such that the result can provide users more candidates, that is, the hot areas with more choices.

7. Conclusion

In this paper, we study the problem of density-based spatial keyword querying. We designed a density based searching algorithm that uses an IR-tree index as the basic indexing structure to achieve a better performance and query effects when considering the density of areas. The experimental results demonstrate that proposed algorithm can return objects in highly clustered areas with high performance. Density based spatial keyword query can be used to support many important information mining services in Web-GIS applications. We will further investigate different density based spatial keyword queries that can provide more intelligent query services over spatial objects with text contents.

Acknowledgments

This work was partially supported by National Science Foundation of China (No. 61075074 and No. 61070183), Natural Science Foundation of Chongqing (No. cstc2012jjB40012), and the Key Discipline Fund of National 211 Project (Southwest University: NSKD11013). The work was also partially supported by State Key Laboratory of Software Engineering (SKLSE) (No. SKLSE2012-09-2).

References

[1] H. Zhuge, Semantic linking through spaces for cyber-physical-socio intelligence: a methodology, Artificial Intelligence 175 (2011) 988–1019.
[2] H. Zhuge, Y. Xing, Probabilistic resource space model for managing resources in cyber-physical society, IEEE Transactions on Services Computing 3 (5) (2012) 404–421.
[3] S.B. Roy, K. Chakrabarti, Location-aware type ahead search on spatial databases: semantics and efficiency, in: Proceedings of SIGMOD 2011, pp. 361–372.
[4] I. De Felipe, V. Hristidis, N. Rishe, Keyword search on spatial databases, in: Proceedings of ICDE 2008, pp. 656–665.
[5] Y.Y. Chen, T. Suel, A. Markowetz, Efficient query processing in geographic web search engines, in: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, ACM, Chicago, IL, USA, 2006, pp. 277–288.
[6] G. Cong, C.S. Jensen, D. Wu, Efficient retrieval of the top-k most relevant spatial web objects, Proceedings of the VLDB Endowment 2 (1) (2009) 337–348.
[7] X. Cao, G. Cong, C.S. Jensen, Retrieving top-k prestige-based relevant spatial web objects, Proceedings of the VLDB Endowment 3 (12) (2010) 373–384.
[8] G. Li, J. Feng, J. Xu, DESKS: direction-aware spatial keyword search, in: Proceedings of VLDB 2012, pp. 824–835.
[9] X. Cao, G. Cong, C.S. Jensen, B.C. Ooi, Collective spatial keyword querying, in: Proceedings of the 2011 International Conference on Management of Data, ACM, Athens, Greece, 2011, pp. 373–384.
[10] J. Lu, Y. Lu, G. Cong, Reverse spatial and textual k nearest neighbor search, in: Proceedings of SIGMOD 2011, pp. 349–360.
[11] R.T. Ng, J. Han, Efficient and effective clustering methods for spatial data mining, in: Proc. 20th Int. Conf. on Very Large Data Bases, Santiago, Chile, pp. 144–155.
[12] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, AAAI Press, Portland, OR, 1996, pp. 226–231.
[13] N. Beckmann, H.P. Kriegel, R. Schneider, B. Seeger, The R*-tree: an efficient and robust access method for points and rectangles, in: Proceedings of SIGMOD 1990, pp. 322–331.
[14] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: Proceedings of SIGMOD 1984, pp. 47–57.
[15] N. Roussopoulos, S. Kelley, F. Vincent, Nearest neighbor queries, in: Proceedings of SIGMOD 1995, pp. 71–79.
[16] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, A. El Abbadi, Constrained nearest neighbor queries, in: Proceedings of Advances in Spatial and Temporal Databases, 2001, pp. 257–276.
[17] G.R. Hjaltason, H. Samet, Distance browsing in spatial databases, ACM Transactions on Database Systems (TODS) 24 (2) (1999) 265–318.
[18] M. Kolahdouzan, C. Shahabi, Voronoi-based k nearest neighbor search for spatial network databases, in: Proceedings of VLDB 2004, pp. 840–851.
[19] Y. Gao, B. Zheng, W.C. Lee, G. Chen, Continuous visible nearest neighbor queries, in: Proceedings of ICDE 2009, pp. 144–155.
[20] J. Zobel, A. Moffat, Inverted files for text search engines, ACM Computing Surveys (CSUR) 38 (2) (2006) 6.
[21] Y. Tao, D. Papadias, Q. Shen, Continuous nearest neighbor search, in: Proceedings of VLDB, 2002, pp. 287–298.
[22] K. Patroumpas, T. Sellis, Monitoring orientation of moving objects around focal points, in: Proceedings of Advances in Spatial and Temporal Databases, 2009, pp. 228–246.
[23] D. Zhang, Y.M. Chee, A. Mondal, A. Tung, M. Kitsuregawa, Keyword search in spatial databases: towards searching by document, in: Proceedings of ICDE 2009, pp. 688–699.
[24] D. Zhang, B.C. Ooi, A.K.H. Tung, Locating mapped resources in web 2.0, in: Proceedings of ICDE 2010, pp. 521–532.
[25] K.W.T. Leung, D.L. Lee, W.C. Lee, Personalized web search with location preferences, in: Proceedings of ICDE 2010, pp. 701–712.
[26] J.B. Rocha-Junior, A. Vlachou, C. Doulkeridis, K. Nørvåg, Efficient processing of top-k spatial preference queries, Proceedings of the VLDB Endowment 4 (2) (2010) 93–104.
[27] B. Yao, F. Li, M. Hadjieleftheriou, K. Hou, Approximate string search in spatial databases, in: Proceedings of ICDE 2010, pp. 545–556.
[28] D. Wu, M.L. Yiu, C.S. Jensen, G. Cong, Efficient continuously moving top-k spatial keyword query processing, in: Proceedings of ICDE 2011, pp. 541–552.
[29] Y. Zhou, X. Xie, C. Wang, Y. Gong, W.Y. Ma, Hybrid index structures for location-based web search, in: Proceedings of CIKM 2005, pp. 155–162.
[30] R. Hariharan, B. Hore, C. Li, S. Mehrotra, Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems, in: Proceedings of SSBDM 2007, p. 16.

Li Zhang is a Ph.D. candidate of Knowledge Grid Research Group, Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences. His research interests mainly focus on database and spatial data query.

Xiaoping Sun is an associative professor of the Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences. His research interests include distributed data management, Semantic Web, P2P computing and decentralized information sharing. His work has been published in many international journals including IEEE TKDE and IEEE TPDS.

Hai Zhuge is a professor of the Key Lab of Intelligent Information Processing, Institute of Computing Technology, Chinese Academy of Sciences. He is the pioneer of Knowledge Grid research and Cyber Physical Society research. He is the author of over 130 papers appeared mainly in leading international journals and conferences such as Artificial Intelligence, Communications of the ACM, IEEE Computer, IEEE Transactions on Knowledge and Data Engineering, IEEE Transactions on Parallel and Distributed Systems, ACM Transactions on Internet Technology, Journal of the American Society for Information Science and Technology and VLDB Conference. He is an ACM Distinguished Scientist, an ACM Distinguished Speaker, and a Senior Member of IEEE. He was the top scholar in the systems and software engineering field during 2000–2004 according to the assessment reports of the Journal of Systems and Software.