Co-spatial Searcher: Efficient Tag-Based Collaborative Spatial Search on Geo-social Network

Jinzeng Zhang[1], Xiaofeng Meng[1], Xuan Zhou[1,2], and Dongqi Liu[1]

[1] School of Information, Renmin University of China, Beijing, China
[2] Key Labs of Data Engineering and Knowledge Engineering, Ministry of Education, China
{zajize,xfmeng,xzhou}@ruc.edu.cn

Abstract. The proliferation of geo-social network, such as Foursquare and Facebook Places, enables users to generate location information and its corresponding descriptive tags. Using geo-social networks, users with similar interests can plan for social activities collaboratively. This paper proposes a novel type of query, called Tag-based top-*k* Collaborative Spatial (*TkCoS*) query, for users to make outdoor plans collaboratively. This type of queries aim to retrieve groups of geographic objects that can satisfy a group of users' requirements expressed in tags, while ensuring that the objects be within the minimum spatial distance from the users. To answer *TkCoS* queries efficiently, we introduce a hybrid index structure called Spatial-Tag R-tree (*STR-tree*), which is an extension of the R-tree. Based on *STR-tree*, we propose a query processing algorithm that utilizes both spatial and tag similarity constraints to prune search space and identify desired objects quickly. Moreover, a differential impact factor is adopted to fine-tune the returned results in order to maximize the users' overall satisfaction. Extensive experiments on synthetic and real datatsets validate the efficiency and the scalability of the proposed algorithm.

Keywords: Spatial collaborative search, Tag, Geo-social network, Shadow prefix-tree.

1 Introduction

With the wide application of location-acquisition technologies, such as GPS, Wi-Fi and Social Networks, Geo-Social Network (*GeoSN*) is increasingly being used in our daily lives. Some examples of *GeoSNs* include Google Buzz, Foursquare, Facebook Places, etc. In a *GeoSN*, a variety of spatial objects (e.g.restaurants, hotels, businesses) are marked on the map and annotated with user generated tags. *GeoSN* users can search for interesting spatial objects, and share information about their location and activities. More importantly, users with similar interests can plan for social activities collaboratively, such as going to somewhere for dining and shopping, or taking a cycling tour together. To make such plans, it is essential to identify a group of spatial objects, such as restaurants, shops and parks, which can maximally satisfy the users' needs.

In this paper, we study how to find suitable spatial objects to meet *GeoSN* users' needs in collaborative activity planning. We formulate a new kind of spatial queries called **Tag-based top-*k* Collaborative Spatial (*TkCoS*) Query**, which aims to retrieve top-k groups of objects for meeting users' needs. In essence, the spatial objects returned

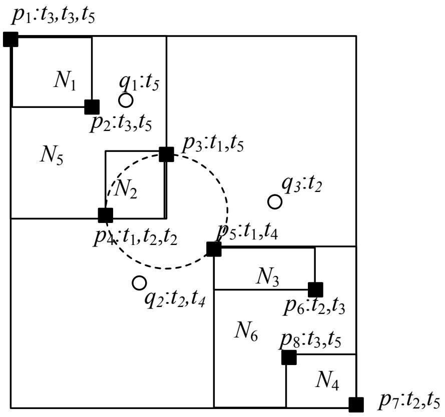


Fig. 1. Example of T*k*CoS query

by a *TkCoS* query should satisfy the following conditions: (1) *they should be annotated with as many tags specified in the query as possible*; (2) *the objects in the result should be as close to one another as possible, such that the maximum diameter of the area covering the objects is minimized*; (3) *the maximum distance between the users' locations and the objects should be minimized.*

Figure 1 illustrates the proposed *TkCoS* query by an example. Points $p_1...p_8$ represent different spatial objects distributed in the region of Los Angeles, each object is annotated with a number of descriptive tags t_i. Suppose three users of a *GeoSN*, Bob, Tom and Mary, plan to find places to meet. They collaboratively submit a *TkCoS* query $Q = \{\langle RowenaAve : ModeratePizza\rangle, \langle MononSt : FreeParking, cinema\rangle, \langle RussellAve : cinema\rangle\}$, where *Rowena Ave($q_1$)* and *Monon St($q_2$)* and *Russell Ave($q_3$)* represent the users' locations, and *Moderate Pizza(t_5), Free parking(t_4), cinema(t_2)* are query tags that express their needs. As shown in Figure 1, the group of objects $\{p_3, p_4, p_5\}$ appears to be the best choice, since it can (1) cover all the user's tags, (2) cover the smallest area, and (3) be near to the users' locations. In contrast, the objects $\{p_2, p_5, p_6\}$ are not suitable. Although they are optimal choices for some individual users (e.g. p_2 for q_1), not all the users' needs are well covered.

TkCoS query can be used in many real world applications. For example, a group of people who want to co-rent a house may have a number of requirements regarding the house's quality, utility, and distance to their working places. These requirements can be met by a single *TkCoS* query. Despite its usefulness,*TkCoS* query poses several challenges to the existing techniques (e.g., [3], [6]) of spatial query processing. Typical spatial keyword queries consider only a single query location (see section 2 for a detailed comparison), and cannot be directly applied to process *TkCoS* queries. A possible adaptation is to use existing (e.g. [6]) methods to execute the sub-queries of a *TkCoS* query separately, and then merge all the results returned by the sub-queries. However, this approach is inefficient. On the one hand , each sub-query has to return a large number of objects to ensure the optimality of the final query results. On the other hand, it will incur high CPU and I/O overheads, as the same data needs to be accessed repeatedly by different queries.

To process *TkCoS* queries efficiently, we devise an efficient hybrid index structure called Spatial-Tag R-tree *(STR)-tree*, which integrates the tag information into a R-tree. To retrieve a group of spatial objects that maximize the users' satisfaction, we propose an algorithm to perform a best-first traversal in the tree. In the algorithm, we employ a *shadow prefix-tree* model to generate candidate sets of search space. An upper bound constraint and a bidirectional constraint are used to prune search space. In addition, we define a differential impact factor to avoid finding the group of objects with covering only a subset of users' requirements. We conduct extensive experiments to evaluate our algorithm using synthetic data sets and real-world data sets. The results demonstrate that the proposed algorithm is efficient and scalable and exhibits superior performance over the brute force method.

The rest of this paper is organized as follows. Section 2 introduces the related work. We formally define the problem of tag-based collaborative spatial search in Section 3. Section 4 introduces the STR-tree. Section 5 introduces our algorithm for processing *TkCoS* queries. Section 6 presents our experimental evaluation. We summarize our work and discuss future work in Section 7.

2 Related Work

In recent years, we have seen an increase in the research dedicated to spatial keyword search. In the query processing of spatial-keyword search, indexing techniques[1],[2],[3],[4],[7] for both text and geographic data are used. Hariharan et al. [1] addressed the problem of spatial keyword queries by utilizing region constraints. Their approach exploits a hybrid index called KR*-tree, which extends R*-tree by augmenting each node with a set of the keywords that appear in the descendants of the node. The query results are the objects located in the query region that are annotated with the query keywords. Felipe et al. [2] proposed a similar kind of query and used IR^2-tree, a combination of R-tree and signature files, to perform query processing. It only contains the information to determine whether a given document contains a query keyword. It is unable to rank the documents based on textual relevance. The work [3] proposes the location-aware top-k text retrieval (L*k*T) query, which takes into account both location proximity and text relevancy. And introduces a hybrid index called IR-tree which integrates R-tree and inverted lists. However, in Web applications, as the number of documents and keywords can be very large, they can result in fat nodes in the IR-trees. The above approaches aim to retrieve only single objects as query results. In contrast, our goal is to find groups of objects such that the objects in a group collectively satisfy the needs of multiple users.

Zhang et al. [4],[5] addressed the problem of m-closest keyword (*m*CK) query. The *m*CK query aims to find the spatially closest tuples which match m user-specified keywords. It utilizes bR*-tree, an integration of R*-tree and bitmap. Each node in the tree is augmented with a keyword MBR to support pruning in the tree traversal. However, as the approach assumes that each object in the result set corresponds to a unique query keyword, it cannot be applied to the cases where multiple constraints are specified on a single object. The work [6] proposes the collective spatial keyword query, aiming to retrieve a group of spatial objects, such that the group's keywords cover the query's keywords and the objects are the nearest ones to the query location. Our *TSkCo* query differs from above approaches in three aspects. First, our query helps multiple users in different locations to search for spatial objects collaboratively. Second, compared to [4][6], our approach aims to find the top-k groups of spatial objects and support partial match of query tags. Third, we exploit vector space model to calculate tag similarity rather than treating all query keywords equally. To summarize, the semantics of *TkCoS* query are different from those of the *m*CK query and collective keyword query.

3 Problem Statement

Let D be a set of spatial objects. Each object in D is represented by a pair o=$\langle loc,\ t\rangle$, where *loc* represents spatial location information and t is a bag of tags for describing the object. In the vector space model of IR[8][9], t can be treated as a vector in finite-dimensional space. This vector can be utilized to calculate the similarity between two sets of tags.

A *TkCoS* query can be represented as Q= $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$, where $q_i.loc$ is the ith user's location and $q_i.t$ represents a set of tags that describe the users' requirements or preferences. The *TkCoS* query intends to retrieve the top-k groups of spatial objects R=$\langle r_1, ..., r_n\rangle$ with the smallest aggregated distance from the users, the minimal spatial coverage and the highest similarity to Q measured in descriptive tags.

In order to search for the top-k best object groups from a spatial dataset, we propose a ranking function to measure how well a search result satisfies a *TkCoS* query. The function takes into account both the spatial proximity and the similarity between tag sets. The spatial proximity, denoted by *D(Q,R)*, can be measured by two components. One is the maximum distance between the sub-query locations of Q and the result set R, denoted by $D_1(Q,\ R)$. The other is the maximum diameter of the area of covering R, denoted by $ODiam(R)$. That is to say,

$$Rank(Q,R) = \alpha \frac{D(Q,R)}{\max D} + (1-\alpha)(1 - Tr(Q.t, R.t)) \tag{1}$$

$$D(Q,R) = \beta D_1(Q,R) + (1-\beta) ODiam(R) \tag{2}$$

$$D_1(Q,R) = \max_{q_i \in Q}\left(\sum_{j=1}^{n} (dist(q_i, r_j))\right) \tag{3}$$

In Formula (1), $Tr(Q.t, R.t)$ denotes the tag similarity between $Q.t$ and $R.t$. $\max D$ denotes the maximal distance between any two objects in D. It is used as a normalization factor. In Formula (3), $dist(q_i,\ r_j)$ denotes the Euclidean distance between an object $r_j \in R$ and a sub-query's location $q_i \in Q$. The parameters $\alpha, \beta \in (0,1)$ are used to adjust the tradeoff between the factors. To measure the maximal diameter of the area covering R, $ODiam(R)$, we give the following definition.

Definition 1. *Given a set of spatial objects* R=$\langle r_1, ..., r_n\rangle$*, the diameter of R, denoted as ODiam.*

$$ODiam(R) = \max_{r_i \in R, r_j \in R} (dist(r_i, r_j)) \tag{4}$$

where $dist(r_i, r_j)$ *measures the Euclidean distance between the two objects* r_i *and* r_j*.*

Compared to a normal document, a tag set usually consists of a much smaller number of terms. Therefore, a direct application of a traditional IR model to measure the tag similarity $Tr(Q.t,R.t)$ in Formula (1) can lead to inadequate results. In this paper, we adopt the method proposed in [10] as our similarity metric, which is defined as follows:

$$Tr(Q.t,R.t) = \sum_{q_i \in Q, r_j \in R} (simt(q_i.t, r_j.t)) \tag{5}$$

$$simt(q_i.t, r_j.t) = \frac{(q_i.t)C(r_j.t)^T}{\sqrt{(q_i.t)C(q_i.t)^T}\sqrt{(r_j.t)C(r_j.t)^T}} \tag{6}$$

In Formula (6), C is a tag similarity matrix, which can be represented by $C{=}(c_{i,j})_{n\times n}$, where n is the number of distinct tags, and $c_{i,j}$ is the similarity value between two tags t_i and t_j.

Finally, the goal of a *TkCoS* query is to find groups of spatial objects with the smallest $Rank(Q,R)$. Our problem can be defined as follows.

Definition 2. *(TkCoS Retrieval). Given a dataset D and a TkCoS query* $Q{=}\ \{\langle q_1.loc, q_1.t\rangle, \ldots, \langle q_m.loc, q_m.t\rangle\}$*, find k groups of objects* $\{R_1, R_2, \ldots, R_k\}$ *(*$R_i{=}\{r_{i1}, r_{i2}, \ldots, r_{in}\}$*), such that there does not exist* $R' \notin \{R_1, R_2, ..., R_k\}$ *that satisfies* $Rank(Q,R') < Rank(Q,R_i)$ *where* $R_i \in \{R_1, R_2, ..., R_k\}$*.*

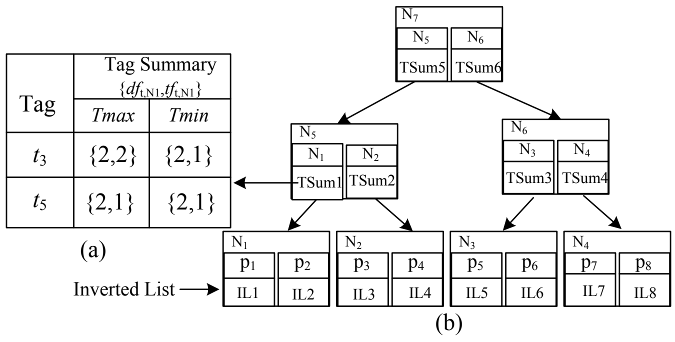
4 STR-Tree: A Refined Hybrid Indexing Mechanism

To answer *TkCoS* queries efficiently, we introduce an efficient hybrid index structure called Spatial-Tag R-tree (*STR-tree*), which is an extension of IR-tree [3] and the original R-tree [11]. It clusters spatially close and semantically relevant objects together and stores the tag information in the nodes of the R-tree [11].

In the *STR-tree*, a leaf node includes entries in the form $(optr, loc, oti)$, where $optr$ is a pointer to an object in D, oti represents the tag information of an object, which is indexed by inverted lists [12]. A intermediate node contains these entries in the form $(Nptr, MBR, Ntsum)$, where *Ntsum* represents the tag summary information of its child nodes referred by *Nptrs*. The *Ntsum* includes two parts: tag maximum information *Tmax* and tag minimum information *Tmin*. Note that each tag in the inverted lists is associated with a tag frequency (*tf*) and the number of objects containing the tag (*df*). To minimize storage overhead, for each tag, the *Tmax* (resp. *Tmin*) of each non-leaf node N_i stores only the *df* and the maximum (resp. minimum) *tf* among all the child nodes rooted at N_i. This maximum (resp. minimum) *tf* provides an upper (resp. lower) bound of the tag similarity between a query and the nodes in the subtree rooted at N_i.

Fig.2 gives an example of *STR-tree* for the spatial objects in Figure 1. Fig. 2(a) shows the *Tmax* and *Tmin* information of the non-leaf node N_1. In Fig. 2(b), the objects p_1 and p_2 are grouped into the node N_1. Likewise, p_3 and p_4 are grouped into N_2. These two non-leaf nodes form a intermediate higher-level node N_5, and so on.

The construction of a *STR-tree* is conducted through a sequence of insert operations, which are a well studied operation in the original R-tree. The only difference is that it needs to update the tag maximal and tag minimal information. Similarly, the update and delete operations of *STR-tree* are simple extensions of those of R-tree too.



Tag	Tag Summary $\{df_{t,N1}, tf_{t,N1}\}$	
	Tmax	*Tmin*
t_3	{2,2}	{2,1}
t_5	{2,1}	{2,1}

Fig. 2. STR-tree indexing structure

5 Processing *TkCoS* Queries

Comparing with other types of spatial keyword queries, a *TkCoS* query is a collaborative query composed of multiple locations associated with multiple tags. A brute force approach is to process each sub-query q_i in Q independently, and merge all the results returned by the sub-queries. Obviously, this approach will lead to high processing cost. First, the same node will be accessed repeatedly in different sub-queries. Second, we need to keep the result set of each sub-query sufficiently large, to ensure the merged results contain the top-k.

In this section, we present a more efficient algorithm to answer *TkCoS* queries. Our idea is to perform a best-first search on the *STR-tree*. When performing the search, we maintain a ranked list of ***candidate node sets***, where each set is a set of the nodes in the *STR-tree* that can potentially contain a top-k result. In each step of the search, we pick the candidate node set with the minimal rank score, and start from its node to traverse the *STR-tree*. Then we use the new nodes encountered in the traversal to form new candidate node sets, and insert them into the ranked list. The candidate node sets are ranked based on the minimum possible score of the results it could contain. During the best-first search, we utilize several pruning strategies to truncate the irrelevant nodes in the *STR-tree*, such that a significant part of the tree can be skipped.

5.1 Query Algorithm

The efficiency of the query algorithm depends on how we evaluate the fitness of each candidate node set. It determines how fast we can reach the bottom of the *STR-tree* and how many irrelevant nodes can be pruned during the search process. To evaluate each candidate node set, we utilize two metrics, that is, the lower bound and the upper bound of the possible scores (defined in Formula (1)) of the results this candidate node set contains. Let *NS* be a candidate node set, and let *Q* be the query, we denote the lower bound and upper bound by *MinRank(Q,NS)* and *MaxRank(Q,NS)* respectively.

Obviously, the lower bound *MinRank(Q,NS)* is used to rank the candidate node sets encountered during the search and prune the paths of the search space in the hybrid index, so as to guarantee that the top-k results returned sequentially.We compute *MinRank(Q,NS)* as follows:

Definition 3. *Given a TkCoS query Q and a node set NS, the minimal possible score of the results in NS w.r.t Q (MinRank) is:*

$$MinRank(Q,NS) = \alpha \frac{MIND_{\varepsilon}(Q,NS)}{\max D} + (1-\alpha)(1 - maxTr(Q.t,NS.u)) \tag{7}$$

$$MIND_{\varepsilon}(Q,NS)) = \beta \max_{n_i \in NS} minDist(Q,n_i) + (1-\beta)(\max_{n_i,n_j \in NS} minDist(n_i,n_j)) \tag{8}$$

where $MIND_{\varepsilon}(Q, NS)$ is the minimal spatial proximity between Q andNS, $\max_{n_i \in NS}$ *$minDist(Q,n_i)$ is the minimal Euclidian distance between Q and NS,* $\max_{n_i,n_j \in NS}$ *minDist (n_i,n_j) is the minimal diameter of NS, $maxTR(Q.t,NS.u)$ is the maximal tag similarity of Q and NS, and α,β and maxD are the same as those in Forumla (1)and (2).*

Lemma 1. ***MinRank(Q, NS)*** *satisfies the following property.*

$$\forall_{os \in Oset} Rank(Q, os) \geq MinRank(Q, NS) \tag{9}$$

where Oset is the spatial object set contained in node NS, os is any subset of Oset.

Proof. First, according to Formula (3), we have *$D_1(Q, os) \geq minDist(Q, NS)$*. Second, according to *Definition 1*, the diameter of a node set is the maximal distance of any pair of its nodes. Thus, we have: ODima(os) > $\max_{n_i,n_j \in NS}$ minDist(n_i,n_j). Third, since *maxTr(Q.t, NS.u)* is the upper bound of tag similarity between *Q* and *NS*, we can *Tr(Q.t, o.t)* $\leq$ *maxTr(Q.t, NS.u)*. We can derive *Rank(Q, os)*$\geq$ *MinRank(Q, NS)* for any *os*. □

Lemma1 proves that *MinRank(Q, NS)* is a true lower bound. Therefore, if we traverse a *STR-tree* in the ascending order of *MinRank(Q, NS)*, we guarantee to find the top-*k* results of *Q*.

The upper bound *MaxRank(Q,NS)* is used to prune the inappropriate candidate node sets as early as possible in search processing. It is calculated as follows.

Definition 4. *Given a TkCoS query Q and a node set NS, the maximum possible score of the results in NS w.r.t Q (MaxRank) is:*

$$MaxRank(Q,NS) = \alpha \frac{MAXD_{\varepsilon}(Q,NS)}{\max D} + (1-\alpha)(1 - minTr(Q.t,NS.l)) \tag{10}$$

$$MAXD_{\varepsilon}(Q,NS)) = \beta \max_{n_i \in NS} maxDist(Q,n_i) + (1-\beta)(\max_{n_i,n_j \in NS} maxDist(n_i,n_j)) \tag{11}$$

where $MAXD_{\varepsilon}(Q, NS)$ is the maximal spatial proximity between Q andNS, $\max_{n_i \in NS}$ *$maxDist(Q,n_i)$ is the maximal Euclidian distance between Q and NS, $minTR(Q.t,NS.l)$ is the minimal tag similarity between Q and NS, and* $\max_{n_i,n_j \in NS}$ *$maxDist(n_i,n_j)$ is the maximal diameter of NS, denoted as* ***maxDima****.*

Lemma 2. ***MaxRank(Q, NS)*** *satisfies the following property.*

$$\forall_{os \in Oset} Rank(Q, os) \leq MaxRank(Q, NS) \tag{12}$$

where Oset is a spatial object set contained in node NS, os is any subset of Oset.

Proof. This lemma can be proved in a similar way as Lemma 1. □

Lemma 3 (Upper Bound Constraint). *Given a TkCoS query Q and a candidate node set NS, Let CNS_k be the kth candidate node set based on the ascending order of MaxRank in the maintained list. The node set NS can be disregarded during traversing the STR-tree if MinRank(Q,NS)> MaxRank(Q,CNS_k).*

Proof. Denoted by *os* the object set enclosed in the node set *NS*. According to Lemma 1, we have: $Rank(Q,os) \geq MinRank(Q,NS)$. As $MinRank(Q,NS) > MaxRank(Q,CNS_k)$, we can derive that $Rank(Q,os) \geq MaxRank(Q,CNS_k)$. Therefore, *NS* cannot contain any top-k results. □

Lemma 2 and 3 proves that $MaxRank(Q,CNS_k)$ is a upper bound(denoted as ***uppC***) of the candidate node sets. Using Lemma 3, we can prune the candidate node set that cannot possibly contain top-k.

In the query processing, apart from considering the ranking function in Formula 1, we should also care the satisfaction degree of each individual user. The objects returned only covering a handful of users' needs should be eliminated from the results. In our work, we adopt *Bayes theory* to define *Contribution Degree* of each sub-query q_i in *Q*.

Definition 5. ***(Contribution Degree.)*** *Given a TkCoS Q=* $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$ *and a node set NS, let* $q_1, q_2, ..., q_m$ *be a partition of Q. Contribution Degree of* q_i *can be defined as follows.*

$$P(q_i|NS) = \frac{P(NS|q_i)P(NS|q_i)}{P(NS)} \tag{13}$$

where $P(q_i)=simt(q_i.t, NS.t)$, $P(NS|q_i)=|q_i.t \cap NS.t|/|NS.t|$, *and* $P(NS) = |Q.t \cap NS.t|/|Q.t|$.

According to Formula (13), when the contribution degree of each sub-query respectively infinitely tend to the proportion of $|\ q_i.t\ |$ in $|\ Q.t\ |$, the users can be maximally satisfied. Therefore, we use the difference between the contribution degree of and q_i and $\frac{|q_i.t|}{|Q.t|}$ to measure degree of satisfaction. We call this difference *Differential Impact Factor.*

Definition 6. ***(Differential Impact Factor)*** *Given a TkCoS Q=* $\{\langle q_1.loc, q_1.t\rangle, ..., \langle q_m.loc, q_m.t\rangle\}$ *and a node set NS, the differential impact factor δ is defined as follows.*

$$\delta_{NS} = \frac{\sqrt{\sum_{i=1}^{m}(P(q_i \mid NS) - \frac{|q_i.t|}{|Q.t|})^2}}{\sqrt{m}max(P(q_i \mid NS) - \frac{|q_i.t|}{|Q.t|})} \tag{14}$$

To take the users' satisfaction degree into account, we use δ_{NS} to modify the lower bound *MinRank(Q, NS)*. Note that the value of δ_{NS} is smaller, the overall satisfaction is better. If $\delta_{NS} \in (0,1)$ is too small, the searching order can be changed obviously. To be specific, we apply $e^{\delta_{NS}}$*MinRank(Q, NS)* to rank the candidate node sets.

Lemma 4 (Bi-directional Constraint). *Given a node set NS and the current node set CNS with the smallest MinRank score, if* $e^{\delta_{NS}}$*MinRank(Q, NS)* $\geq$ $e^{\delta_{CNS}}$*MinRank(Q, CNS), then the node set NS is pruned.*

Proof. Obvious from Lemma 1 and definition 3 and 5. □

In order to find top-*k* groups of spatial objects, *STR-tree* is traversed from the root node following the best-first traversal strategy. The pseudocode is shown in Algorithm 1. Let a min-priority queue *U* keep track of candidate node sets *E* with e^{δ_E}*MinRank(Q, E)* , while an ordered link list *LL* store the same nodes in *U* associated with *MaxRank(Q,E)* and the *maxDima(E)* in the ascending order of *MaxRank*. The process iteratively checks the first entry *E* in *U*(line 4-23). If *E* contains only spatial objects, it is returned as a top-k result. Otherwise, If *E* is a intermediate node set, we invoke Algorithm 2 to the children of the nodes in *E*, and compose them into new candidate node sets S_{nl} (line 11).

Algorithm 1: COSS(*Q*, *STR-tree*, *k*)

Input: *Q*: a *TkCoS* query;
STR-tree: a hybrid indexing;
Output: The top-k groups of objects satisfying *Q*;

```
1:  $U \leftarrow$ new min-priority queue; $LL \leftarrow$ new a ordered link list;
2:  $U$.Enqueue($STR\text{-}tree.root, 0$); $LL$.Insert($STR\text{-}tree.root, \infty, \infty$);
3:  $uppC \leftarrow \infty$; $uppDima \leftarrow \infty$;
4:  while $U$ is not empty do
5:     $E \leftarrow U$.Dequeue(); $LL$.Delete($E$);
6:     $uppC \leftarrow$ LL[k]; $uppDima \leftarrow$ max($maxDima(LL[1..k])$);
7:     if $E$ is a group of objects then
8:       $R \leftarrow R \cup \{E\}$;
9:       if $|R| = K$ then goto 25;
10:    else if $E$ is a intermediate nodeset then
11:          $S_{nl} \leftarrow$ GenCSet($E, uppC, uppDima$);
12:          for each nodeset $NS$ in $S_{nl}$ do
13:              if $|U$.length$| < k$ or $MinRank(NS, Q) < uppC$ then
14:                 $U$.Enqueue($NS$, $e^{\delta_{NS}} MinRank(NS, Q)$);
15:                 $LL$.Insert($NS$, $MaxRank(NS, Q), maxDima(NS)$);
16:                 $uppC \leftarrow LL$[k]; $uppDima \leftarrow$ max($maxDima(LL[1..k])$);
17:          else if $E$ contains leaf nodes then
18:             $S_l \leftarrow$ GenCSet($E, uppC, uppDima$);
19:             for each objectset $os$ in $S_l$ do
20:                 if $MinRank(os, Q) <$ uppC then
21:                    $U$.Enqueue($os$, $e^{\delta_{os}} Rank(Q, os)$);
22:                    $LL$.Insert($os$, $MaxRank(os, Q), maxDima(os)$);
23:                     $uppC \leftarrow$ LL[k]; $uppDima \leftarrow$ max($maxDima(LL[1..k])$);
24:  return $R$
```

Then, we consider each of the new node sets. If the node set *NS* in S_{nl} does not satisfy the condition in Lemma 3, it is enqueued to *U* together with $e^{\delta_{NS}}$*MinRank(Q, NS)* and is inserted *LL* with *MaxRank(Q,NS)* and *maxDima(NS)*. Otherwise, *NS* will be discarded, because it cannot contain any top-k. Whenever *LL* changes, we need to update *uppC*, which represents *k*-th smallest *MaxRank(Q,NS)* in *U*, and *uppDima* that is the maximal *maxDima* of top-*k* element in *LL*(line 16). Likewise, if E is a leaf node set, we process E in the same way to the non-leaf nodes (line 17-23). The algorithm repeats the above procedure. Once *R* contains *k* groups of objects or no more groups of objects can be found, the algorithm terminates and outputs *R*.

5.2 Generating Candidate Node Sets of Search Space

During each step of the best-first search algorithm, it needs to expand the nodes in a candidate node set, and use their child nodes to generate more concrete candidate node sets. An efficient generation approach is essential to ensure the efficiency of the top-k algorithm. However, if we exhaustively enumerate all the subsets, it could incur high computing overhead, as the number of subsets grows exponentially with the number of child nodes. In order to reduce the cost of I/O and computation, we need to filter out irrelevant node sets as early as possible. We exploit the *apriori* property among the set and its superset to reduce search space in generating candidate node sets. By using the

```
Algorithm 2: GenCset(S, uppC, uppDima)
Input: S: a set of spatial nodes;
Output: A list of candidate spatial node sets SList;
1:  $Slist \leftarrow \emptyset$; $T \leftarrow$ SPF-tree(S,uppDima);
2:  for each node $n_i$ in S do
3:      for each childnode $cn_i$ in $n_i$ do
4:          if $cn_i.t \cap Q.t \neq \Phi$ and $\alpha MIND_{\varepsilon}(cn_i,Q) < uppC$ then
5:              $I_1 \leftarrow I_1 \cup cn_i$;
6:  for k form 2 to |Q.t| do
7:      $NN_k \leftarrow$ GenNeighbor($I_{k-1}$, T);
8:      for each nodeset NS in $NN_k$ do
9:          if $\alpha MinDist_{\varepsilon}(NS,Q) < uppC$ then
10:             $I_k \leftarrow I_k \cup NS$;
11:     $L \leftarrow \cup_k I_k$;
12: for each nodeset $NS \in L$ do
13:     if (MinRank(NS, Q) < uppC) then
14:         add NS to SList;
15: return SList;
Procedure GenNeighbor($I_{k-1}$, SPF-tree)
16: for each nodeset l in $I_{k-1}$ do
17:     PreOrderTraverse(SPF-tree);
18:     for each node $n_i$ in l do
19:         $CS_{ni} \leftarrow$ Get-Childnodeset($n_i$);
20:         $CommonCS \leftarrow \cap_{ni} CS_{ni}$;
21:     for each node CN in CommonCS do
22:         $C \leftarrow Merge(n_i,CN)$; add C to $NN_k$;
23: return $NN_k$;
```

upper bound of candidate node sets *uppC* and the upper bound of the diameter *uppDima* introduced in section 5.1, we devise two pruning mechanisms to filter out the candidate node sets that cannot possibly contain any top-k result.

Lemma 5. *Given a TkCoS query Q and a node set NS, if* $\alpha MIND_{\varepsilon}(NS,Q) > uppC$*, then the node set NS and all its supersets cannot contain any top-k result.*

Proof. According to definition 3, we have $MinRank(Q,NS) = \alpha\frac{MIND_{\varepsilon}(Q,NS)}{\max D} + (1-\alpha)(1-maxTr(Q.t,NS.u)$. On the one hand, the minimal spatial proximity of a superset of *NS* is larger than $MIND_{\varepsilon}(Q,NS)$. On the other hand, the tag similarity *maxTr(NS,Q)* is in the range between 0 and 1. If we set *maxTr(NS,Q)* to 1, then $\alpha MIND_{\varepsilon}(Q,NS)$ is the lower bound of the *MinRank* of all its superset. Therefore, when $\alpha MIND_{\varepsilon}(NS,Q)$ $>$ *uppC* holds, any superset of *NS* has larger *MinRank* score than the scores of the current top-k candidate node sets (because *uppC* is a upper bound). Thus *NS* and all its supersets cannot contain any top-k result. □

By applying Lemma 5 to the generation of the candidate node sets, the node sets that cannot affect the query results can be discarded as early as possible. We call the node set that does not satisfy the condition in Lemma 5 *Relevant Node Set*, denoted as ***I***. Besides, we still utilize the the upper bound of diameter *uppDima* for pruning.

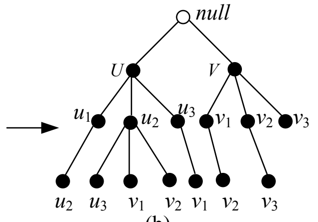
Lemma 6. *Given a node set NS=*$\langle N_1,\ldots,N_k\rangle$*, if the diameter of NS is larger than uppDima where uppDima is the maximal maxDima of top-k element in link list, then NS and its superset can be pruned.*

Proof. According to definition 1, if the diameter of *NS* is larger than *uppDima*, then there exists two nodes $N_i,N_j \in N$ with $minDist(N_i,N_j) \geq uppDima$. Any superset of *NS* must contain N_i,N_j and its diameter exceeds the *uppDima*. Thus *NS* and its superset does not provide a query result with a diameter less than *uppDima*. □

Lemma 6 says that the diameter of the candidate node set can not exceed *uppDima*. In generating candidate node sets, we only care these node sets with neighbor relationship that the distance of any two nodes is less than *uppDima*. Once any two nodes in *NS* satisfy neighbor relationship, we call it a *Neighbor Node Set*, denoted as ***NN***.

U_c	u_1u_2; $u_2u_3v_1v_2$; u_3v_1
V_c	v_1v_2; v_2v_3; v_3

(a)



(b)

Fig. 3. The construction of *shadow prefix-tree*

In the sequel, we proceed to propose the strategy of generating candidate node sets. A good candidate generation method keeps the aprior properties, as well as avoids amounts of join operations. Based on this principle and lemma 6, we propose a shadow prefix-tree model that materializes the neighbor relationship between childnodes of *NS*. The number of subtree is determined by the number of nodes in *NS* while each branch in subtree records the neighbor relationship of childnodes of *NS*. Figure4 illustrate an example of shadow prefix-tree about node set $(U(u_1,u_2,u_3),V(v_1,v_2,v_3))$. we can find the neighbor node set by traversing the shadow prefix-tree according to lemma 7.

Lemma 7. ***(Neighbor Node Set Generation)*** *Given a relevant node set* $I_{k-1}=\{n_1,n_2,...,n_{k-1}\}$, *if a node* n_k *is contained in the intersection of child nodes of each node in* I_{k-1}, $NN_k=\{n_1,n_2,...,n_k\}$ *is a size k neighbor node set.*

Proof. Each node n_i in $I_{k-1}=\{n_1,n_2,...,n_{k-1}\}$ has neighbor relationship with other node of I_{k-1}. If a node n_k is the child node of $\{n_1,n_2,...,n_{k-1}\}$, then indicates that n_k has a neighbor relationship with all nodes in I_{k-1}. In addition, the neighbor relationship is symmetric. So $NN_k=\{n_1,n_2,...,n_k\}$ is a size-k neighbor node set. □

Algorithm 2 shows the process of generating candidate node sets. In this algorithm, NN_k represents size-k neighbor node sets, I_k is the size-k relevant node sets. The shadow prefix-tree T is firstly built by utilizing S and the upper bound of diameter *uppDima* (line 1). Then, we invoke procedure GenNeighbor (line 16-23) to generate the neighbor node sets NN_k based on T and node set I_{k-1}. I_k can be obtained by filtering the node sets in NN_k that satisfy the condition of lemma 5 (line 9-10). Finally, all of node sets in I_k are appended to list L(line 11). After all node sets L are found, we check each node set in L to see whether its *MinRank* score is less than *uppC* (line 12-14). Those that cannot qualify the conditions are eliminated. On contrary, we do not check this constraint in the process of node set I_k since if a node set does not contain query tags, it can still combine other nodes to covering the missing tags. As long as it is relevant to the tag of query, we keep it in list L of node set I_k.

6 Experiments

This section presents an extensive experimental evaluation of the proposed method for *TkCoS* queries using synthetic and real datesets.

6.1 Experimental Setting

We used two **Baseline Algorithms** to compare with our proposed algorithm **COSS**.
Baseline1 First separate last union (FSLU). In FSLU, we process each subquery q_i separately using an existing spatial keyword query processing method proposed in [2]. We utilize *STR-tree* to retrieve the object set with smallest rank score for each q_i. Then we merge the results of the sub-queries to obtain the final top-k.

Baseline2 Centroid-based Iterative Search Algorithm (CISA). In CISA, we use an aggregation centroid c to substitute for a *TkCoS* query Q. The tag information of the centroid c can be considered as the tags combination of each subquery in Q. The query processing iteratively utilizes the methods of [2] to retrieve the group of objects that cover all tags and are nearest to centroid c. It firstly finds the object with the smallest distance that matches a part of tags. The uncovered tags together with the location of centroid *c* form a new query. This process terminates until the tags are either matched or skipped (for there is no matching object).

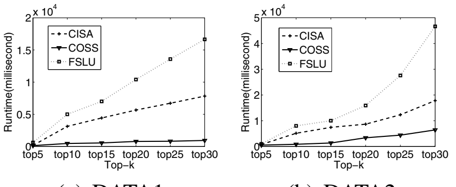
Datasets and Queries. Our experiments used two datasets, whose properties are summarized in Table 1.The real data set DATA1 is obtained from the online web data resource PocketGPSWorld [13] that consists of points of interest locations in United Kingdom. DATA2 is a synthetic dataset generated to simulate a geo-social application. We extracted 3000,000 tags from del.ic.ious [14] and combined the tags with the real spatial dataset about California's streets to generate DATA2. We generated 5 query sets for DATA1, each containing 2,4,8,16,32 tags respectively. Similarly, we generated 5 query sets for DATA2. Each of the query sets comprises of 50 queries and each query is randomly generated. Based on the query sets, we generate 6 group query sets for each dataset. The number of the sub-queries in each group query ranges from 2 to12.

We implement all the algorithms using VC++. In all the experiments, the index structures were disk-resident and the page size was fixed at 4KB. In an index, the number of children in each node is determined solely by the size of a page. All our experiments were executed on a Windows platform with an Intel(R) Core(TM)2 Duo CPU of T7500 @ 2.66GHZ and 4GB RAM.

Table 1. Shows more details of the two datasets

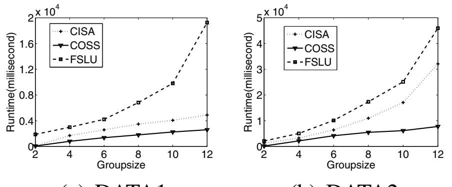
Dataset	Total # of objects	Total # of unique tags	Total # of tags
DATA1	125,313	47,672	877,191
DATA2	2,249,727	289,175	8,998,908

6.2 Performance Evaluation



(a) DATA1 (b) DATA2

Fig. 4. Effect of the *k* value



(a) DATA1 (b) DATA2

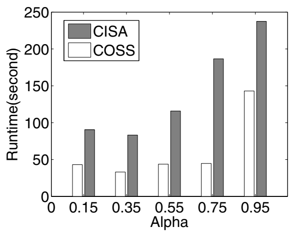
Fig. 5. Effect of the group size

We compare our algorithm COSS against FSLU and CISA in answering *TkCoS* queries. The running time is used as our performance metric. We conduct four sets of experiment in total.

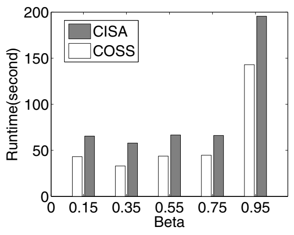
Effect of k. In this set of experiments, we evaluate the performance of the three algorithms with a varying *k*. As shown in Fig.4(a) and 4(b), the COSS method notably outperforms FLSU and CISA for all values of k. Meanwhile, CISA performs better than FSLU. This is mainly because COSS can prune irrelevant nodes more effectively than the other two methods. As expected, the running time of all the approaches increases with increasing k.

Effect of the Group Size of Query. The objective of our second set of experiments is to study the efficiency of the three algorithms in dealing with different sizes of query groups. The results on DATA1 and DATA2 are shown in Fig.5(a) and 5(b). We can see that COSS significantly outperforms both FSLU and CISA. For all the approaches, the query running time increases as group size grows. This is because when increasing group size, it takes more time to process the increasing number of entries in the hybrid index. Nevertheless, the growth rate for COSS is much smaller than the others.

Effect of α and β. Fig.6(a) shows the performance of CISA and COSS on DATA1 with respect to different α. It is clear that COSS significantly outperforms CISA for all the values of α. Recall that α can adjust the importance between the spatial proximity and the tag similarity. A larger α means that the spatial distance is more important, while a smaller α means that the tag information is more important. We notice that the running time increases as α increases. This is mainly because spatial proximity is normally less selective in pruning irrelevant results. The impact of the parameter β on the performance of CISA and COSS algorithm is shown in Fig.6(b). As mentioned earlier, β is introduced to balance the importance of the distance between query Q and results R and that of the covering area of R. We obverse that COSS also outperforms CISA slightly in most cases.



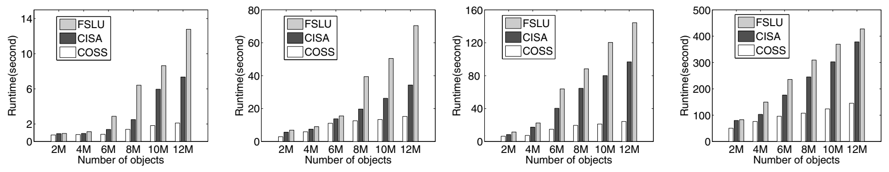
(a) α



(b) β

Fig. 6. Effect of α and β

Scalability in Terms of Dataset Size. In order to simulate the real geo-social networking in which the number of objects and tags continuously increasing, our final set of experiments is conducted to evaluate the scalability of three algorithms by varying the number of objects. We increase the size of the synthetic dataset steadily from 2 million to 12 million. Fig.7 shows the running time of the algorithms as the data size increases. When the group size is small, the CISA and COSS shows the similar rate of increase in running time. As group size grows, CISA's running time increases more dramatically. We can also see that all the algorithms scale smoothly when the number of objects is not greater than 4 million. However, the performance of FLSU and CISA declines quickly when the dataset size is above 4 million. On the contrary, our COSS method scales well even with large dataset size.



(a) groupsize=2 (b) groupsize=4 (c) groupsize=6 (d) groupsize=8

Fig. 7. Scalability in terms of dataset size

7 Conclusions

In this paper, we study the problem of tag-based top-k collaborative spatial (*TkCoS*) query , which aims to find groups of objects with the smallest rank score and the highest satisfaction degree for multiple users. We present an efficient query processing algorithm that is based a hybrid index, STR-tree and employ upper bound constraint and bi-directional constraint to prune irrelevant subtree. This algorithm tackles the key challenge by building a shadow prefix tree model to generate candidate search space. Our experimental evaluation shows that the proposed algorithm is efficient and scalable and superior performance compared with baseline method. Our results can be used as a value-added service in today's social networking websites or geo-based applications.

Acknowledgments. This research was partially supported by the grants from the Natural Science Foundation of China (No. 61070055, 60833005, 91024032, 91124001), the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University of China (No. 12XNLJ01, 11XNL010, 10XNI018), National Science and Technology Major Project (No. 2010ZX01042-002-003).

References

1. Hariharan, R., Hore, B., Li, C., Mehrotra, S.: Processing spatial keyword (sk) queries in geographic information retrieval systems. In: 19th IEEE International Conference on Scientific and Statistical Database Management, pp. 161–170. IEEE Press, Washington (2007)
2. Felipe, I.D., Hristidis, V., Rishe, N.: Keyword search on spatial databases. In: 24th IEEE International Conference on Data Engineering, pp. 656–665. IEEE Press, Washington (2008)
3. Cong, G., Jensen, C.S., Wu, D.: Efficient retrieval of the top-k most relevant spatial web objects. J. Proc. of VLDB Endowment 2(1), 337–348 (2009)
4. Zhang, D.X., Chee, Y.M., Mondal, M., Tung, A.K., Kitsuregawa, M.: Keyword search in spatial databases: Towards searching by document. In: 25th IEEE International Conference on Data Engineering, pp. 688–699. IEEE Press, Washington (2009)
5. Zhang, D.X., Ooi, B.C., Tung, A.K.H.: Locating mapped resources in web 2.0. In: 26th IEEE International Conference on Data Engineering, pp. 521–532. IEEE Press, Washington (2010)
6. Cao, X., Cong, G., Jensen, C.S.: Collective spatial keyword querying. In: 31th ACM International Conference on Management of Data, pp. 373–384. ACM Press, New York (2011)
7. Khodaei, A., Shahabi, C., Li, C.: Hybrid Indexing and Seamless Ranking of Spatial and Textual Features of Web Documents. In: Bringas, P.G., Hameurlain, A., Quirchmayr, G. (eds.) DEXA 2010. LNCS Part I, vol. 6261, pp. 450–466. Springer, Heidelberg (2010)

8. Wong, S.K.M., Ziarko, W., Raghavan, V.V.: On modeling of information retrieval concepts in vector space. ACM Transaction on Database System 12(2), 299–321 (1987)
9. Anh, V.N., de Kretster, O., Moffat, A.: Vector space ranking with effective early termination. In: ACM 24th International Conference on Research and Development in Information Retrieval, pp. 35–42. ACM Press, New York (2001)
10. Park, J., Choi, B.C., Kim, K.: A vector space approach to tag cloud similarity ranking. J. Information Processing Letters 110(12-13), 489–496 (2010)
11. Guttman, A.: R-Trees: A dynamic index structure for spatial searching. In: 4th ACM International Conference on Management of Data, pp. 47–57. ACM Press, New York (1984)
12. Zobel, J., Moffat, A.: Inverted files for text search engines. ACM Computing Surveys 38(2), 6 (2006)
13. PocketGPSWorld, `http://www.pocketgpsworld.com/modules.php?name=POIs`
14. Delicious, `http://www.delicious.com/`