POSTER

Spatial-aware interest group queries in location-based social networks

YAFEI LI, Hong Kong Baptist University, Hong Kong, Hong Kong

DINGMING WU, Hong Kong Baptist University, Hong Kong, Hong Kong

JIANLIANG XU, Hong Kong Baptist University, Hong Kong, Hong Kong

BYRON KOON KAU CHOI, Hong Kong Baptist University, Hong Kong, Hong Kong

WEIFENG SU

PDF Download
2396761.2398713.pdf
27 January 2026
Total Citations: 7
Total Downloads: 347

Published: 29 October 2012

Citation in BibTeX format

CIKM'12: 21st ACM International Conference on Information and Knowledge Management
October 29 - November 2, 2012
Hawaii, Maui, USA

Conference Sponsors:
SIGIR
SIGWEB

Spatial-aware Interest Group Queries in Location-based Social Networks

Yafei Li[†] Dingming Wu[†] Jianliang Xu[†] Byron Choi[†] Weifeng Su[§]
[†]Department of Computer Science, Hong Kong Baptist University, xujl@comp.hkbu.edu.hk
[§]Division of Science and Technology, United International College, yafeili@uic.edu.hk

ABSTRACT

Location-based social networks, such as Foursquare and Facebook Places, are bridging the gap between the physical world and online social networking services through acquired user locations. Some social networks released check-in services that allow users to share their visiting locations with their friends. In this paper, users' interests are modeled by check-in actions. We propose a new spatial-aware interest group (SIG) query that retrieves a user group of size k where every user is highly interested in the query keyword and also spatially close to each other. An efficient algorithm AIR based on the IR-tree is proposed for the processing of SIG queries. Furthermore, an optimization is developed and achieves a much better performance than the baseline algorithm.

Categories and Subject Descriptors

H.2.4 [**Database Management**]: Query Processing

General Terms

Algorithms, Performance

Keywords

Query Processing, Spatial Databases, Social Networks

1. INTRODUCTION

Location-based social networks, such as Foursquare and Facebook Places, are bridging the gap between the physical world and online social networking services through acquired user locations. Some social networks released check-in services that allow users to share their visiting locations with their friends. These locations considered as spatial objects are associated with some tags that describe their features, e.g., spatial object 'starbucks' with tags 'food', 'beverage', and 'coffee'. If a user checks in the spatial object 'starbucks', the user may be interested in 'food', 'beverage', or 'coffee'. Such voluntary check-in actions reflecting the users' interests can be used for marketing. As an example, a firm may want to find a location to promote its product, such that some users close to that location are interested in the product.

In this paper, users' interests are modeled by check-in actions. We propose a new kind of query that retrieves a user group of size k where every user is highly interested in the query keyword and also spatially close to each other. A sample query may request a user group of size 3 such that the three users are fond of 'movie' and close to each other. We call this type of query a *Spatial-aware Interest Group* (SIG) query. It contains a query keyword and the size k of the requested user group. The answer to the query is a user group of size k that maximizes a ranking function combining the diameter (distance between the farthest pair of users) of the group and the group interest of the query keyword.

We propose an efficient algorithm AIR based on the IR-tree for the processing of SIG queries. An optimization is further developed to effectively prune the search space. Experiments based on a real dataset show that the proposed algorithms achieve up to 2 orders of magnitude improvement over the baseline algorithm.

2. PROBLEM DEFINITION

Let $\mathcal{D}$ be a set of spatial objects. Each spatial object p is associated with a set of tags $p.\Gamma$. Let $\mathcal{U}$ be a set of users. Each user $u \in \mathcal{U}$ is a triple (id, λ, ν), where id is the user's identification, λ is the user's home location, and ν is a vector of the user's interests of the tags that are associated with the spatial objects checked in by the user. The interest value of a tag is defined in Definition 1.

Definition 1. Let D_u be the set of spatial objects checked in by user u and let D_t be the set of spatial objects that are associated with tag t. Function $Count(u, p)$ counts the number of check ins of a user u to the spatial object p.

The interest value for tag t of user u is computed as:

$$I(u,t) = \frac{\sum_{p \in D_u \wedge p \in D_t} Count(u,p)}{\sum_{p \in D_u} Count(u,p)}. \quad (1)$$

Table 1 shows an interest vector of a user. Higher value indicates more interest. As an example, the user is more interested in 'sport' than 'movie'.

Table 1: Sample Interest Vector

Tag	movie	sport	music	supermarket
Interest Value	0.10	0.20	0.36	0.14

We derive a ranking function as a weighted sum of normalized terms for ranking a user group G_k of size k with respect to a query q, denoted by $rank_q(G_k)$:

$$rank_q(G_k) = \alpha \frac{I(G_k, q.t)}{I_{max}(q.t)} + (1-\alpha)(1 - \frac{D(G_k)}{D_{max}}), \quad (2)$$

where $q.t$ is the query keyword that belongs to the tag space of the dataset, $I(G_k, q.t)$ is the group interest of the query keyword $q.t$, defined as the minimum value of the user interest in the group, i.e., $I(G_k, q.t) = \min\{I(u, q.t) \mid u \in G_k\}$, $D(G_k)$ is the diameter of group G_k, i.e., the Euclidean distance between the farthest pair of users in the group, $D(G_k) = \max\{||u_i.\lambda\ u_j.\lambda|| \mid u_i, u_j \in G_k\}$, where $||u_i.\lambda\ u_j.\lambda||$ is the Euclidean distance between two users. The maximum group interest $I_{max}(q.t)$, i.e., the k^{th} largest user interest of tag $q.t$, and the maximum group diameter D_{max}, i.e., the distance between the farthest pair of users in the *whole* dataset, are used for normalization. Parameter $\alpha \in [0, 1]$ is used to balance the group interest and the group diameter.

A Spatial-aware Interest Group (SIG) query consists of two components: a keyword $q.t$ and the size k of the requested user group. It retrieves a user group of size k, such that the ranking function (Eq. (2)) is maximized.

Figure 1 illustrates an example SIG query with three different values of α in the ranking function (Eq. (2)). The circles, squares, and triangles in the figure depict the locations of a set of users. Given an SIG query q, the sizes of those shapes indicate the user interests of the query keyword $q.t$. The bigger the size, the higher the user interest. Query q requests a user group of size 4 that maximizes the ranking function. The solid circles are the result group when $\alpha = 0$, i.e., only the group diameter is considered. The solid squares are the result group when $\alpha = 0.5$. The solid triangles represent the query result when $\alpha = 1$, i.e., only the group interest is considered.

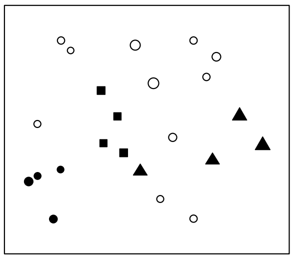


Figure 1: Example SIG Query

Problem Statement: We study the efficient processing of SIG queries. We aim for a solution that has short response time.

3. BASELINE QUERY ALGORITHM

We next discuss how to exploit existing techniques for processing SIG queries. While no baseline algorithm exists for SIG queries, we propose an approach that uses an inverted file [6] to compute group interests and adopts an R-tree [3] to compute group diameters separately. Specifically, users' interests of all tags are pre-computed and indexed by an inverted file, where the posting list of each tag is sorted in descending order of users' interests. Users' home locations are indexed by an R-tree.

Given an SIG query, let set $\mathcal{G}_k$ contain all possible user groups of size k. For each user group $G_k \in \mathcal{G}_k$, we derive an upper bound on the ranking score of G_k, as shown in Theorem 1.

THEOREM 1. *Let D_{min} be the minimum group diameter among all possible groups, i.e., $D_{min} = \min\{D(G_k) \mid G_k \in \mathcal{G}_k\}$. An upper bound on the ranking score of a user group G_k is*

$$rank^u_q(G_k) = \alpha \frac{I(G_k, q.t)}{I_{max}(q.t)} + (1-\alpha)(1 - \frac{D_{min}}{D_{max}}). \quad (3)$$

The proof of Theorem 1 is obvious and thus omitted. The value of D_{min} can be determined as follows. We first retrieve the closest pair of users u_i and u_j, and then check whether the scribing circle of u_i and u_j covers another $k-2$ users. If yes, the distance between u_i and u_j is returned as D_{min}. Otherwise, the next closest pair of users is retrieved, and the process is repeated.

The basic idea of the baseline algorithm is that user groups are processed in descending order of the upper bounds on their ranking scores. If the ranking score of the current found user group is higher than the upper bound on the ranking score of the next user group, the current found user group is returned as the result, i.e., the ranking function is maximized. Since the group interest $I(G_k, q.t)$ is the minimum value of the user interests in G_k, based on Eq. (3), the descending order of the upper bounds can be guaranteed by processing user groups in descending order of the user interest. If two user groups have the same group interest, the user group with smaller group diameter is processed first. Algorithm 1 shows the pseudo code of the baseline algorithm. It starts from the user u_i with the k^{th} largest interest, and finds a group of size k that contains u_i, denoted by $G_k(u_i)$ (Line 4). The group $G_k(u_i)$ returned by GetNextGroup$(rtree, u_i, k)$ (Algorithm 2) satisfies (i) the interest of u_i is the smallest and (ii) the diameter of $G_k(u_i)$ is the minimum. The result is found when the ranking scores of unprocessed groups cannot exceed the ranking score of the current group (Line 7).

Algorithm 1 Baseline(Int k, Keyword t, InvertedFile $invf$, RTree $rtree$)

```
1: $pl \leftarrow$ load the posting list of $t$ from $invf$;
2: Result $G_k \leftarrow \emptyset$;
3: for $i$ from $k$ to $pl.length$ do;
4:     $G_k(u_i) \leftarrow$ GetNextGroup$(rtree, u_i, k)$;
5:     if $rank_q(G_k(u_i)) > rank_q(G_k)$ then
6:         $G_k \leftarrow G_k(u_i)$;
7:     if $rank_q(G_k) > rank^u_q(G_k(u_{i+1}))$ then
8:         Return $G_k$;
9: Return $G_k$;
```

4. ADVANCED QUERY ALGORITHMS

We now present an advanced algorithm based on the IR-tree [2] (AIR) for the processing of SIG queries. To further improve the performance of AIR, we develop an optimization that involves a group diameter constraint. The group diameter constraint is effective for search space pruning.

4.1 AIR Algorithm

We adopt the IR-tree [2] to index users, where users' home locations are represented as the locations of objects and users' interests are as the documents of objects. Based on the IR-tree, we propose an efficient version of function GetNextGroup (Algorithm 2), named AIRGetNextGroup (Algorithm 3). Function AIRGetNextGroup takes advantage of

Algorithm 2 GetNextGroup(RTree *rtree*, User u_i, Int k)

```
1: $Queue \leftarrow$ NewPriorityQueue();
2: $Queue$.Enqueue($rtree.root, 0$);
3: Add $u_i$ to $G'_k$;
4: while $Queue$ is not empty do
5:     Entry $e \leftarrow Queue$.Dequeue();
6:     if $e$ refers to a user then
7:         if the interest of $e >$ the interest of $u_i$ then
8:             Add $e$ to $G'_k$;
9:             if $G'_k$ contains more than $k$ users then
10:                $G_k \leftarrow$ select the group of size $k$ with the minimum diameter from $G'_k$;
11:                if $D(G_k) \leq ||u_i \;\; e||$ then
12:                    Return $G_k$;
13:    else
14:        for each entry $e'$ in the node pointed to by $e$ do
15:            $Queue$.Enqueue($e', ||u_i \;\; e'||$);
16: Return $G'_k$;
```

Algorithm 3 AIRGetNextGroup(RTree *rtree*, User u_i, Int k)

```
1: $Queue \leftarrow$ NewPriorityQueue();
2: $Queue$.Enqueue($rtree.root, 0$);
3: Add $u_i$ to $G'_k$;
4: while $Queue$ is not empty do
5:     Entry $e \leftarrow Queue$.Dequeue();
6:     if $e$ refers to a user then
7:         if the interest of $e >$ the interest of $u_i$ then
8:             Add $e$ to $G'_k$;
9:             if $G'_k$ contains more than $k$ users then
10:                $G_k \leftarrow$ select the group of size $k$ with the minimum diameter from $G'_k$;
11:                if $D(G_k) \leq ||u_i \;\; e||$ then
12:                    Return $G_k$;
13:    else
14:        for each entry $e'$ in the node pointed to by $e$ do
15:            if the interest of $e' >$ the interest of $u_i$ then
16:                $Queue$.Enqueue($e', ||u_i \;\; e'||$);
17: Return $G'_k$;
```

the IR-tree that each entry in each node has an upper bound on the user interests contained in the subtree pointed to by the entry. If the interest of an entry e is smaller than the interest of a user u, no user in the subtree of the entry e can have a larger interest than does the user u, and thus the subtree can be pruned (Line 15). The AIR algorithm adopts Algorithm 1, but calling AIRGetNextGroup instead of GetNextGroup at Line 4.

4.2 Optimization

In the baseline and the AIR algorithms, user groups are processed in descending order of their group interests. Let $G_k(u_i)$ be the current found user group. A condition of the next processed user group $G_k(u_{i+1})$ being the query result can be presented in Theorem 2.

THEOREM 2. *$rank_q(G_k(u_i)) < rank_q(G_k(u_{i+1})) \iff D(G_k(u_{i+1})) < D_c$, where*

$$D_c = D_{max}\left(1 - \frac{rank_q(G_k(u_i)) - \alpha\frac{I(G_k(u_{i+1}),q.t)}{I_{max}(q.t)}}{(1-\alpha)}\right). \quad (4)$$

The proof is obvious and easily derived based on Eq. (2), and thus omitted. Theorem 2 tells that given two user groups $G_k(u_i)$ and $G_k(u_{i+1})$ and the group interest of $G_k(u_i)$ is higher than that of $G_k(u_{i+1})$, if the group diameter of $G_k(u_{i+1})$ is smaller than D_c, the ranking score of $G_k(u_{i+1})$ is higher than that of $G_k(u_i)$. Thus, in the optimized algorithm, the group diameter constraint D_c is used to prune the search space (see Algorithms 4 and 5). The value of D_c is updated when a user group with a higher ranking score is found (Line 8 in Algorithm 4). When processing the next user group (Algorithm 5), only the user group with diameter smaller than D_c is considered (Line 16). Otherwise, it is impossible to have a ranking score higher than the current found user group (guaranteed by Theorem 2). Hence, some user groups are pruned and the computation cost is reduced.

Algorithm 4 AIRStar(Int k, Keyword t, InvertedFile *invf*, RTree *rtree*)

```
1: $pl \leftarrow$ load the posting list of $t$ from $invf$;
2: Result $G_k \leftarrow \emptyset$;
3: $D_c \leftarrow \infty$;
4: for $i$ from $k$ to $pl.length$ do;
5:     $G_k(u_i) \leftarrow$ AIRStarGetNextGroup($rtree, u_i, k, D_c$);
6:     if $rank_q(G_k(u_i)) > rank_q(G_k)$ then
7:         $G_k \leftarrow G_k(u_i)$;
8:         Update $D_c$ according to Eq. (4);
9:     if $rank_q(G_k) > rank^u_q(G_k(u_{i+1}))$ then
10:        Return $G_k$;
11: Return $G_k$;
```

Algorithm 5 AIRStarGetNextGroup(RTree *rtree*, User u_i, Int k, Double D_c)

```
1: $Queue \leftarrow$ NewPriorityQueue();
2: $Queue$.Enqueue($rtree.root, 0$);
3: Add $u_i$ to $G'_k$;
4: while $Queue$ is not empty do
5:     Entry $e \leftarrow Queue$.Dequeue();
6:     if $e$ refers to a user then
7:         if the interest of $e >$ the interest of $u_i$ then
8:             Add $e$ to $G'_k$;
9:             if $G'_k$ contains more than $k$ users then
10:                $G_k \leftarrow$ select the group of size $k$ with the minimum diameter from $G'_k$;
11:                if $D(G_k) \leq ||u_i \;\; e||$ then
12:                    Return $G_k$;
13:    else
14:        for each entry $e'$ in the node pointed to by $e$ do
15:            if the interest of $e' >$ the interest of $u_i$ then
16:                if $||u_i \;\; e'|| < D_c$ then
17:                    $Queue$.Enqueue($e', ||u_i \;\; e'||$);
18: Return $G'_k$;
```

5. EXPERIMENTS

We evaluate the performance of the three algorithms for the processing of SIG queries, including the baseline, the AIR algorithm, and the AIR with optimization (AIR*).

5.1 Dataset and Queries

We use a real dataset crawled from jiepang.com, which is a popular location-based social network in China. Table 2 lists the properties of the dataset.

For the sake of scalability test, we generate 6 datasets of size varied from 50,000 to 300,000 within the space of the real dataset. One query set containing 200 queries is

Table 2: Dataset Properties

Total # of users	353,493
Total # of spatial objects	244,331
Total # of check-in actions	5,250,466
Total # of unique tags	2,101
Average # of tags per spatial object	2
Average # of tags per user interest	1.3

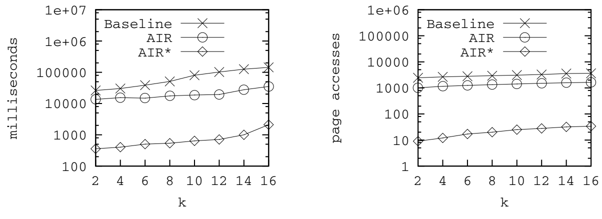
randomly generated. We report the average elapsed time and the average I/O cost of the three algorithms.

5.2 Setup

The indexes, the R-tree, the inverted file, and the IR-tree used in this paper are disk resident. The page size is set to 4KB. The fanouts of the R-tree and the IR-tree are both 200. All the algorithms are implemented by Java programming language. The models of the CPU and RAM are AMD Athlon Dual Core Processor 2.8G Hz and 2GB DDR2 memory. The default values of k and α are 10 and 0.5, respectively.

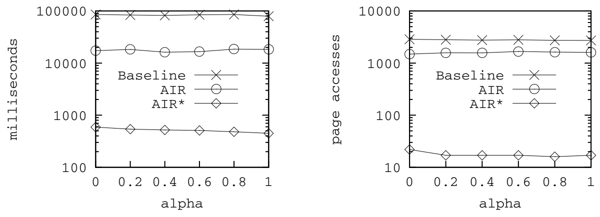
5.3 Performance Results

Varying k. Figures 2(a) and 2(b) show the average elapsed time and the average simulated I/O cost when varying the group size k. Both the elapsed time and the I/O cost increase as k grows. The AIR and AIR* algorithms outperform the baseline approach for all values of k in terms of both metrics, since the IR-tree is able to prune the search space according to both the group interests and group diameters. Notably, the AIR* algorithm achieves 2 orders of magnitude improvement due to the effective use of the group diameter constraint.



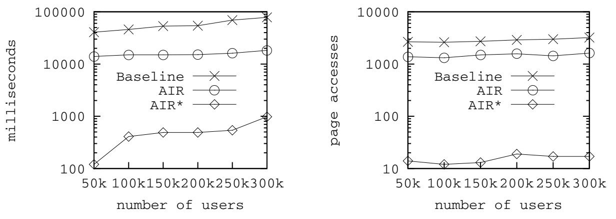
(a) Elapsed Time (b) I/O Cost
Figure 2: Varying k

Varying α. Parameter α is used to balance the group interest and the group diameter. Only the group interest is considered when $\alpha = 1$. The group diameter dominates the ranking function when α approaches 0. Figures 3(a) and 3(b) show the performance of the three algorithms with different values of α. The AIR* algorithm again beats the other two algorithms by 2 orders of magnitude in terms of both metrics.



(a) Elapsed Time (b) I/O Cost
Figure 3: Varying α

Varying the Size of Dataset. In Figures 4(a) and 4(b), the average elapsed time and the average simulated I/O cost increase sub-linearly as the size of dataset grows. The AIR* algorithm significantly outperforms the baseline and AIR in all cases tested.



(a) Elapsed Time (b) I/O Cost
Figure 4: Varying the Size of Dataset

6. RELATED WORK

There is a stream of research on spatial query processing in the literature. Recently, spatial queries have been extended to incorporate text keywords [1, 2, 5]. Cong *et al.* [2] presented a new indexing scheme, IR-tree, for location-aware top-k object retrieval. Zhang *et al.* [5] studied an m-closest-keyword (mCK) query that finds a set of spatially closest objects covering m specified keywords. Cao *et al.* [1] proposed a collective spatial keyword query that retrieves a group of nearby spatial objects to collectively cover the specified keywords. Unlike these previous queries, the proposed SIG query explores the relationship between users' locations and interests in the specified keywords.

Group queries have also been studied in the context of social networks. In [4], Yang *et al.* proposed a social-temporal group query to find a group of activity attendees with the minimum total social distance to the query issuer. In contrast, our group query is more concerned with the common interest of group members rather than their social relationships.

7. CONCLUSIONS

In this paper, we have identified a new SIG query arising in location-based social networks. We have proposed an efficient algorithm based on the IR-tree, namely AIR, for the processing of SIG queries. An optimization has also been developed for AIR to further prune the search space. The experiments based on a real dataset have demonstrated that the optimized AIR* algorithm achieves 2 orders of magnitude improvement over the baseline algorithm.

Acknowledgements. This work is supported by RGC GRF 211212, HKBU FRG2/11-12/074, and NSFC61073017.

8. REFERENCES

[1] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. *SIGMOD*, pages 349–360, 2011.

[2] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[3] A. Guttman. R-trees: A dynamic index structure for spatial searching. *SIGMOD*, pages 47–57, 1984.

[4] D.-N. Yang, Y.-L. Chen, W.-C. Lee, and M.-S. Chen. On social-temporal group query with acquaintance constraint. *PVLDB*, 2011.

[5] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. *ICDE*, 2009.

[6] J. Zobel and A. Moffat. Inverted files for text search engines. *ACM Comput. Surv.*, 38(2), 2006.