

Retrieving Top-k Famous Places in Location-Based Social Networks

Ammar Sohail^{1(✉)}, Ghulam Murtaza², and David Taniar¹

¹ Faculty of Information Technology, Monash University, Melbourne, Australia
`{ammr.sohail,david.taniar}@monash.edu`

² Intersect, Melbourne, Australia
`ghulam@intersect.org.au`

Abstract. The widespread proliferation of location-acquisition techniques and *GPS*-embedded mobile devices have resulted in the generation of geo-tagged data at unprecedented scale and have essentially enhanced the user experience in location-based services associated with social networks. Such location-based social networks allow people to record and share their location and are a rich source of information which can be exploited to study people's various attributes and characteristics to provide various Geo-Social (GS) services. In this paper, we propose a new type of query called *Top-k famous places T_kFP* query, which enriches the semantics of the conventional spatial query by introducing a social relevance component. In addition, three approaches namely, (1) Social-First (2) Spatial-First and (3) Hybrid are proposed to efficiently process *T_kFP* queries. Finally, we conduct an exhaustive evaluation of the proposed schemes using real and synthetic datasets and demonstrate the effectiveness of the proposed approaches.

1 Introduction

The fusion of social and geographical information has given rise to the notion of online social media known as location-based social networks (LBSNs) such as Facebook and Foursquare. An LBSN is usually represented as a complex graph where nodes represent various entities in the social network (such as users, places or pages) and the edges represent the relationships between different nodes. These relationships are not only limited to friendship relations but also contain other types of relationships such as **works-at**, **born-in**, and **studies-at** etc. In addition, the nodes may also contain spatial information such as a user's check-ins at different locations. Consider the example of a Facebook user Alice who was born in Germany, works at Monash University and checks-in at a particular restaurant. Facebook records this information by creating Facebook pages for *Monash University* and *Germany* and adding their relationships with the node corresponding to Alice [1], e.g., Alice and Monash University are connected by an edge labelled **works-at** and Alice and Germany are connected with an edge labelled **born-in**. The check-in information records the places the user has visited.

Spatial data and social relationships in LBSNs provide a rich source of information which can be exploited to offer many interesting services. Consider the example of a German tourist visiting Melbourne. She may want to find a nearby pub which is popular (e.g., frequently visited) among people from Germany. This involves utilizing spatial information (i.e., near by pub, check-ins) as well as social information (i.e., people who were **born-in** Germany). Similarly, a user may want to find near by places that are most popular among her friends, e.g., the places most frequently visited by her friends.

Although various types of queries have been studied on LBSNs [2–7], to the best of our knowledge, none of the existing techniques can be applied to answer the queries like the above that aim at finding near by places that are popular among a particular group of users satisfying a social constraint. Motivated by this, in this paper, we formalize this problem as a *Top-k famous places* T_kFP query and propose efficient query processing techniques. Specifically, a T_kFP query retrieves top- k places (points of interest) ranked according to their spatial and social relevance to the query user where the spatial relevance is based on how close the place is to a given location and the social relevance is based on how frequently it is visited by the one-hop neighbors of the query user in the social graph. A formal definition is provided in Sect. 2.1.

We present three approaches to answer T_kFP query processing called, (1) Social-First, (2) Spatial-First and (3) Hybrid. The first two approaches separately process the social and spatial components of the query and do not require a specialized index. The third approach (*Hybrid*) is capable of processing social and spatial components simultaneously by utilizing a hybrid index specifically designed to handle T_kFP queries.

We make the following contributions in this paper.

1. To the best of our knowledge, we are the first to study the T_kFP queries that retrieves near by places popular among a particular group of users in the social network.
2. We propose three approaches to process the query which enable flexible data management and algorithmic design.
3. We conduct an exhaustive evaluation of the proposed schemes using real and synthetic datasets and demonstrate the effectiveness of the proposed approaches.

2 Preliminaries

2.1 Problem Definition

Location Based Social Network (LBSN): A *location-based social network* consists of a set of entities U (e.g., users, Facebook Pages etc.) and a set of places P . The relationship between two entities u and v are indicated by a labeled edge where the label indicates the type of relationship (e.g., **friend**, **lives-in**). LBSN also records check-ins where a check-in of a user $u \in U$ at a particular place $p \in P$ indicates an instance that u had visited the place P .

Score of a place p : Given a query user q , and a range r , the score of a place $p \in P$ is 0 if $\|q, p\| \geq r$ where $\|q, p\|$ is the Euclidean distance between query location and p . If $\|q, p\| \leq r$, the score of p is a weighted sum of its spatial score (denoted as $p_{spatial}$) and its social score (denoted as p_{social}).

$$p.score = \alpha \times p_{spatial} + (1 - \alpha) \times p_{social} \quad (1)$$

where α is a parameter used to control the relative importance of spatial and social scores. The social score p_{social} is computed as follows. Let F_q denote the one-hop neighbors of the query user considering a particular relationship type, e.g., if the relationship is **born-in** and the query entity is the Facebook Page named Germany, then F_q is a set of users born in Germany. Although our techniques can be used on any type of relationship, for the ease of presentation, in the rest of the paper we only consider the friendship relationships. In this context, F_q contains the friends of the query user q . Let $p.visitors$ denote the set of all users that visited (i.e., checked-in at) the place p . The social score p_{social} is computed as follows.

$$p_{social} = \frac{|F_q \cap p.visitors|}{|F_q|} \quad (2)$$

where $|X|$ denote the cardinality of a set X . Intuitively, p_{social} is the proportion of the friends of q who have visited the place p .

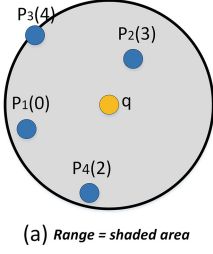
The spatial score $p_{spatial}$ is based on how close the place is to the query location. Formally, given a range r , $p_{spatial} = 0, (r - \|q, p\|)$ where $\|q, p\|$ indicates Euclidean distance between the query location and p . Note that p_{social} is always between 0 to 1 and we normalize $p_{spatial}$ such that it is also within the range 0 to 1, e.g., the data space is normalized such that $\|q, p\| \leq 1$ and $r \leq 1$.

Top-k Famous Places (T_kFP) Query: Given a LBSN, a T_kFP query q returns k places with the highest scores where the score $p.score$ of each place p is computed as described above.

Example 2.1: Fig. 1(a) illustrates the locations of a set of places $P = \{p_1, p_2, p_3, p_4\}$. The query q shown in Fig. 1(a) with $k = 2$ and range $r = 0.15$, has a set of friends $F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$. The number in bracket next to each place is the check-in count made by q 's friends. Figure 1(b) shows the Euclidean distances and the visitors of each place amongst q 's friends. Let us assume $\alpha = 0.5$, the score of the p_1 w.r.t. q is computed as $0.025 + 0 = 0.025$. Similarly, we have $Score(p_2) = 0.185$, $Score(p_3) = 0.205$ and $Score(p_4) = 0.115$. The result of the query q is (p_2, p_3) according to scoring function in Eq. 1.

2.2 Related Work

Since Geo-Social query processing is an emerging field, there is only limited literature available about it [3, 8–10]. In [11], an Algebraic model is proposed to process geo-social query. This model consists of set of operators to query the geo-social data. They replicate the graphs to represent spatial and social components



P	$ q, p_i $	Pi.visitors	Final Score
P1	0.10	-	0.025
P2	0.08	u1,u3,u9	0.185
P3	0.14	u5,u6, u2, u7	0.205
P4	0.12	u7, u10	0.115

(b) $\alpha = 0.5, k = 2, r = 0.15$
 $F_q = \{u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}\}$

Fig. 1. Top- k Query example

which can be very large in terms of social networks thus, making query processing more cumbersome. Huang et al. [12] studied a Geo-Social query that retrieves the set of nearby friends of a user that share common interests, without providing concrete query processing algorithms. In [6], they defined a new query namely, *Geo-Social Circle of Friends* to retrieve the group of friends in geo-social settings whose members are close to each other based on their geographical and social circumstances such as for group sports, social gathering and community services. Yang et al. [7] introduced another type of group query by extending the work presented in [6] namely, *Social-Spatial Group Query* (SSGQ) which is useful for impromptu activity planning.

In addition, nearest neighbour queries have been widely applied in location-based social networks recently [13–17]. Wu et al. [18] proposed a new query named as social-aware *top-k* spatial keyword query (SkSK) which retrieves a list of k objects ranked according to their spatial proximity, textual (e.g. restaurant has menu and different facilities) and social relevance. Another work is presented by Jiang et al. [19] in which they proposed a method to find *top-k* local users in geo-social media data. First, they compute spatial score of a user who has posted tweets in given region followed by computing social score of her tweets based on number of replies or forwards to her tweets. Subsequently, by fusing both scores, the user is ranked accordingly.

To the best of our knowledge, none of the existing algorithms can be applied or trivially extended to answer T_kFP queries studied in this paper.

2.3 Framework Overview

The proposed framework consists of three approaches to answer T_kFP query: (I) Social-First (II) Spatial-First and (III) Hybrid. The *Social-First* approach first processes the social component (e.g., friendship relations and their check-ins) and then processes the spatial component (e.g., places in given range) where as *Spatial-First* initially processes the spatial component followed by processing the social component. In contrast, the *Hybrid approach* is capable of processing both social and spatial components simultaneously to answer such queries. More specifically, it leverages two types of pre-processed information associated

with each user $u \in U$, her check-in information (check-ins) and summary of her friends' check-ins information.

To the best of our knowledge, there is no unanimously accepted social or spatial storage implementation. Specifically, Facebook uses adjacency lists stored in Memcached [20] which is a distributed memory caching system, on the other hand, Twitter leverages the R*-Tree [21] spatial index. Further, Foursquare uses MongoDB [22], a document oriented database. Similarly, academics research has been adopting various kind of approaches such as [5] uses adjacency list stored in Neo4j which is a graph based database, where as [11] utilizes relational tables for storing the friendship relations.

Similar to the existing work on KNN queries, we tailored the storage implementation for our technique in a way that suites our requirements. More specifically, we index places and users' check-in information by adopting the R-Tree [23] spatial Index structure. Before presenting our technique, we present the definition of *Facility R-Tree*, *Check-in R-Tree* and *Friendship Index*.

Facility R-Tree: Stores all places ($p \in P$) in a given data set. A node of *Facility R-Tree* is represented by a minimum bounded rectangle (MBR) that constitutes all places in its sub-tree.

Check-in R-Tree: For the sake of efficiency, we store the check-in information of each user by indexing all visited places by her in a separate index based on R-Tree.

Friendship Index: To index each user $u \in U$ and their social relationships, we build an index structure by employing *B⁺-Tree* based on *Unicorn* [1] built to search the *facebook* social graph.

3 Proposed Technique

3.1 Social-First Based Approach

Intuitively, *Social-First* approach, first processes the social component of given query q and computes the score of all the places $p \in P$ which have been checked-in by her friends $u \in F_q$. Next, using given range r , it processes the spatial component of the q and computes the score of the remaining places $p \in P$ which are not checked-in and returns the set of *top-k* places based on their score and q 's defined preference criteria α . We next describe the technique in detail with pseudocode given in Algorithm 1.

Initially, in the first loop of the algorithm, we compute the score of each place p in given range r , that has been visited by the friends $u \in F_q$ of query q while maintaining the score of current k_{th} place p based on their social and spatial scores by exploiting the *Check-in R-Tree* of each friend u . In-addition, in the second loop, the *Facility R-Tree* is exploited to compute the score of those places $p \in P$ in range r which are not visited by q 's friends hence, their respective score only comprises of spatial component and their $(P_{social}) = 0$ which subsequently yields the *top-k* result set. Let us assume, the score of current k_{th} place p is

$Score_k$, next lemma shows that if the $\|q, p\| \geq (r - \frac{Score_k}{\alpha})$, we can prune that place p . Next, we introduce our first pruning rule in Lemma 1.

Lemma 1. *Every place p that has a distance $\|q, p\|$ from query q greater than current $(r - (Score_k/\alpha))$, cannot be in the Top- k places.*

Proof. Given a query user q , a range r , preference factor α , a place p which is not checked-in by any user $u \in F_q$ has social component (p_{social}) = 0, by using Eqs. 1 and 2 we get,

$$Score(p) = \alpha(r - \|q, p\|) + 0 \quad (3)$$

To be the candidate for the Top- k places, a place p 's score must be greater than current $Score_k$, hence

$$Score_k \leq Score(p) \quad (4)$$

By substituting the value of $Score(p)$ from Eq. 3,

$$\begin{aligned} Score_k &\leq \alpha(r - \|q, p\|) \\ \|q, p\| &\leq (r - (Score_k/\alpha)) \end{aligned} \quad (5)$$

□

Algorithm 1. Social-First

Input : Query q , range r , weight-age constant α , integer k ,
Output: Result set R

```

1 foreach friend  $u$  in  $F_q$  do
2   | Traverse check-in R-Tree of  $u$  ; // Accessing Check-in R-Tree by branch
   | and bound method
3   | foreach place  $p$  in  $r$  in Check-in R-Tree do
4   |   | update social score and score of  $p$ ;
5   |   | update  $Score_k$ ;
6   | end
7 end
8 Traverse Facility R-Tree ; // Accessing Facility R-Tree by branch and
   | bound method
9 foreach place  $p$  in range  $r$  in Facility R-Tree do
10  | if  $p.dist \leq (r - (Score_k/\alpha))$  ; // from Lemma 1
11  | then
12  |   | compute  $Score(p)$ ;
13  |   | update  $Score_k$ ;
14  | end
15 end
16 return Return  $R$ 

```

3.2 Spatial-First Based Approach

Initially, this approach starts with the processing of spatial component of the query q by computing the score of each place p in given range r regardless of the fact whether it is checked-in by any friend $u \in F_q$ of q or not. Moreover, it then computes the social score of each place p by first computing the number of friends checked-in to it by performing an intersection of the set of q 's friends F_q and the set of visitors of the place V_p followed by computing the social score p_{social} of p and then yields the result set. We next elaborate the technique in detail with pseudocode given in Algorithm 2.

Specifically, for each place $p \in P$ in range r , we compute the $Score(p)$ in ascending order of the distance of the place p from q . To achieve this, a *Heap* is initialized with the root entry of *Facility R-Tree* with $\|q, e\|$ as a key to process spatial component first. Each entry is iteratively retrieved from *Heap* and processed as follows. For each place p in range r , we first, compute social component of the score by counting the number of friends $u \in F_q$ who have visited the place p followed by the computation of the final score using Eq. 1. Let us assume, the score of current k_{th} place p is $Score_k$, next lemma shows that if the $\|q, p\| \geq (r - \frac{(Score_k - (1 - \alpha))}{\alpha})$, the process stops since every subsequent place p entry in *Heap* is further than the current place p entry from q . Next, we introduce our second pruning rule in Lemma 2.

Lemma 2. *Every place p that has distance $\|q, p\|$ from query q greater than current $Score_k$, cannot be in the Top-k places.*

Proof. Given a query user q , a range r , preference factor α , to be the candidate for the Top-k places, a place p 's score must be greater than current $Score_k$, by using Eqs. 1 and 2, we get,

$$Score_k \leq Score(p) \quad (6)$$

By substituting the value of $Score(p)$, we get,

$$Score_k \leq \alpha(r - \|q, p\|) + (1 - \alpha) \left(\frac{|F_q \cap V_p|}{|F_q|} \right) \quad (7)$$

Since the maximum possible social score of given place can be 1, we get

$$\begin{aligned} Score_k &\leq \alpha(r - \|q, p\|) + (1 - \alpha) * 1 \\ \|q, p\| &\leq (r - \frac{(Score_k - (1 - \alpha))}{\alpha}) \end{aligned} \quad (8)$$

□

3.3 Hybrid Approach

Friends Check-Ins R-Tree: To Optimize, we propose a spatial indexing structure, the *Friends Check-ins R-tree*, that supports the simultaneous pruning of

Algorithm 2. Spatial-First

Input : Query q , range r , weight-age constant α , integer k ,
Output: Result set R

```

1 Traverse Facility R-Tree ;      // Accessing Facility R-Tree by branch and
  bound method
2 foreach place  $p$  in  $r$  in Facility R-Tree do
3   if  $\|q, p\| \geq (r - ((Score_k - (1 - \alpha))/\alpha))$  ;      // from Lemma 2
4   then
5     | return Result set  $R$ 
6   end
7   count friends  $\leftarrow F_u \cap V_p$ ;
8   compute  $Score(p)$ ;
9   update  $Score_k$  ;
10 end
11 return Result set  $R$ 

```

friends and places. It is an R-Tree based structure which is constructed for each user $u \in U$ and is able to prune the search space. *FCR-Tree* stores check-in information of each friend $u \in F_q$ of q , thus representing the check-in summary of all friends of q . The objects of *FCR-Tree* are the root MBRs of each friend's *Check-in R-Tree*. The update of the index in case of new check-in entry of any friend u , is not costly since these are being bulk updated after certain period of time.

Let us assume a query $q \in U$ where the friends of q are $F_q = \{u_2, u_3, u_4, u_5, u_6\}$. Figure 2 shows the conceptual view of the *FCR-Tree* of u_1 . In following Sect. 3 we describe our proposed technique in detail.

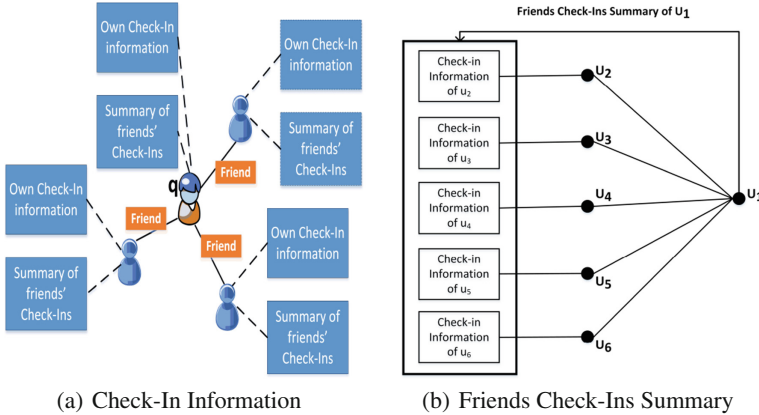


Fig. 2. Check-Ins Example

In this approach, the score of each place p in range r is computed by processing the social and spatial components of query q together. To answer the T_kFP queries efficiently, this approach leverages the *Friends Check-ins R-Tree* of query q to prune the friends which have not visited the top-k places and *Grid Spatial Index* to prune the places in given range r which cannot be the candidate for the top-k result set. More specifically, to compute the social and spatial scores of a place p , this approach supports simultaneous pruning of q 's friends set F_q and places $p \in P$ in given range r . We next elaborate the technique in detail with pseudocode given in Algorithm 3.

To achieve this, initially, grid portioning approach is employed to divide the area formed by given range r into small cells. Similarly, for each grid cell g_c , a set of places $P_{g_c} \in P$ which lie inside the cell is maintained by using the *Facility R-Tree* and distance of the closest place p to q in a cell is recorded as the cell distance from q . In addition, a set of friends who might have visited a cell denoted as V_{cell} is computed for each cell by exploiting *Friends Check-ins R-Tree (FCR-Tree)* of q and counting the number of overlapping objects of *FCR-Tree* with the cell. Figure 3 illustrates an example of a cell and overlapping objects in which four objects are overlapping with the cell and therefore, the overlap count of the cell is 4.

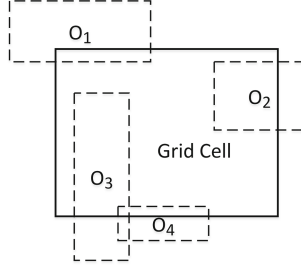


Fig. 3. Cell Overlap

Once the overlap count and distance to the q for each cell g_c is computed, a ranking score of each cell is computed using Eq. 9 which serves as an upper bound on the score of any place in the cell. Moreover, to compute the score of a place p in range r , the places $p \in P_{g_c}$ of the cell g_c with highest score are processed first. If the current k_{th} score is greater than the next cell's score, the process stops since all subsequent cells can not contain a place with higher ranking score.

$$Score_{cell} = \alpha(r - cell.distance) + (1 - \alpha) \left(\frac{OverlapCount_{cell}}{|F_q|} \right) \quad (9)$$

Algorithm 3. Hybrid Algorithm

Input : Query q , range r , weight-age constant α , integer k ,
Output: Result set R

```

1 Traverse Facility R-Tree ; // Using branch and bound method
2 foreach place  $p$  in  $r$  in Facility R-Tree do
3   compute  $Score(p)$ ;
4   insert  $p$  in corresponding cell's places set;
5   update the distance of corresponding cell;
6   update  $Score_k$ ;
7 end
8 Traverse FCR-Tree of  $q$  ; // Using branch and bound method
9 foreach cell  $g_c$  do
10  compute  $overlapCcount(g_c)$  and  $Score(g_c)$ ;
11 end
12 foreach cell  $g_c$  do
13   if  $Score_{g_c} \leq Score_k$  then
14     return Result set  $R$ ;
15   end
16   foreach place  $p$  in cell  $g_c$  do
17     compute check-in count of  $p \leftarrow V_{cell} \cap V_p$ ;
18     update  $Score(p)$  and  $Score_k$ ;
19   end
20 end
21 return Result set  $R$ 
```

4 Experiments

4.1 Experimental Setup

To the best of our knowledge, there is no prior algorithm to solve T_kFP queries therefore, we compare the three proposed algorithms with each other to evaluate their performance.

All algorithms were implemented in C++ and experiments were run on Intel Core I3 2.4 GHz PC with 8GB memory running on 64-bit Ubuntu Linux. Specifically, we use same real data set of *Gowalla* as of [24]. *Gowalla* is a location based social network which later was acquired by *Facebook*. It contains 196,591 users, 950,327 friendships, 6,442,890 check-ins and 1,280,956 checked-in places across the world. The page size of each *Facility R-Tree* index is set to 4096 Bytes and 1024 Bytes for *Check-in R-Tree* and *FCR-Tree* indexes. We randomly select 100 users and treat them as query points. The cost in the experiments correspond to the average cost of 100 T_kFP queries. The default value of range r is 100 Km and the default value of k is set to 10 unless mentioned otherwise.

4.2 Performance Evaluation

Effect of Range: We analyse the performance of our algorithms for various range values ranging from 10 – 400 kms. The size of the area formed by given

range determines the number of places it contains (ranging from 1500 – 94000). Figure 4 shows that *Spatial-First* algorithm is most affected at bigger range values hence its performance deteriorates due to large number of places. Note that, the *Hybrid* algorithm performs better for bigger range since it is more likely to find *top-k* places after processing fewer cells. Figure 4(b) shows that I/O cost increases for bigger range values due to large number of places in range which result in higher index access rate.

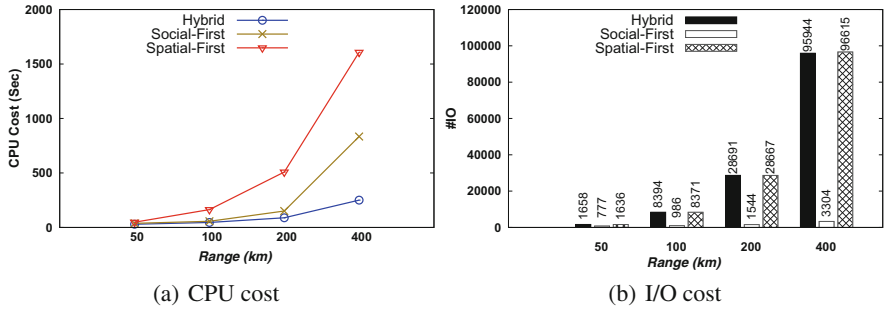


Fig. 4. Effect of varying range (number of places)

Effect of Average number of Friends: In Fig. 5, we study the effect of the average number of friends of each query. Note that the size of *FCR-Tree* depends on the size of friends set of each user in data set which essentially affects the Hybrid algorithm. Further, *Spatial-First* algorithm is greatly affected by it because the intersection of two large sets i.e. visitors set and friends set is more expensive. Specifically, Fig. 5(a), shows the CPU cost and Fig. 5(b) shows the I/O cost of each method for varying average number of friends. The average number of places in given range r is 38319. Note that when average number of friends increases, the CPU and I/O cost of all three algorithms increases since each friend’s check-in information is required to verify candidate places.

Effect of concurrent number of Queries: Geo-Social services seek to answer large number of incoming queries simultaneously due to the enormous size of registered users. Therefore, the number of concurrent queries ranging from 50 to 200 are analyzed for all three algorithm. In addition, each experiment involves average number of friends ranging from 750 – 1350 and approximately 10,000 average number of places in given range r . All three algorithms need to traverse the *Facility-RTree* every time a T_kFP query is issued to verify candidate place. The *Social-First* algorithm also needs to traverse the *Check-in R-Tree*. On the other hand, *Hybrid* algorithm leverages the *FCR-Tree* and both *Spatial-First* and *Hybrid* greatly rely on the visitors set of the places. In Fig. 6, we report the CPU and I/O cost of each algorithm on *Gowalla* data set for different number of queries. As expected, the I/O cost of *Social-First* algorithm is less than the other two due to low dependency on indexes. Note that *Hybrid* is up to four times better than *Social-First* and *Spatial-First* algorithms.

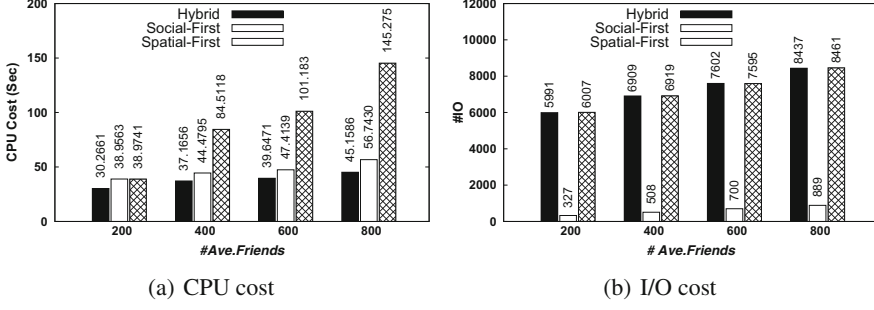


Fig. 5. Performance comparison on different number of friends

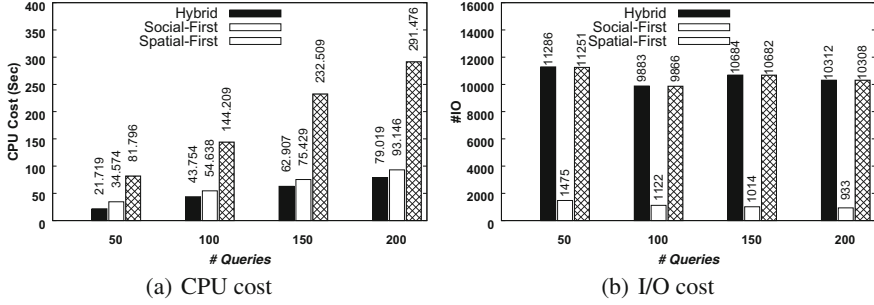


Fig. 6. Effect of number of Queries

Effect of Grid Size: In Fig. 7(a), we study the effect of the size of grid partitioning ranges from 2 – 64 on Hybrid algorithm. The size of grid affects the CPU cost since the size of a cell defines how many places will be processed/pruned at once. Similarly, it also affects the termination condition on the algorithm. Note that the best CPU performance can be achieved by dividing the area into grid of size 4×4 .

Effect of k : In previous experiments, the value of k is set to 10. Next, we analyze the performance of three algorithms for various values of k . Note that in Fig. 7(b), all three algorithms are nearly independent of k . The reason is that, we have to update the result set every time we update the score of a place. Therefore, the size of the result set does not impose great computation load. In terms of I/O cost, Fig. 7(c) shows that all three algorithms do not get affected by the value of k .

Effect of Data Set Size: In Fig. 8(a) and (b), we study the effect of data set size on the performance of the three algorithms. Specifically, we conduct experiments on synthetic data sets of different sizes containing places ranging from 100k to 500k. In Fig. 8(a), note that the *Spatial-First* algorithm is most effected by number of places. Similarly, in Fig. 8(b), *Hybrid and Spatial-First* have higher I/O cost due to the intersection performed on visitors set of places and friends set of query q .

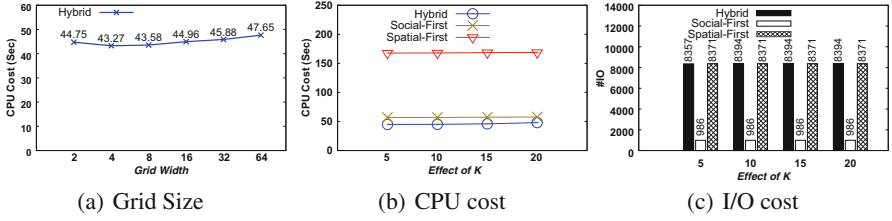


Fig. 7. Effect of Grid Size and varying number of requested places (k)

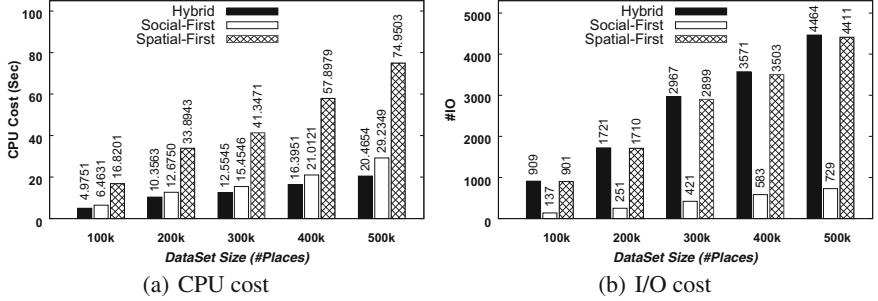


Fig. 8. Effect of varying data set sizes (number of places)

5 Conclusions

This paper studies *Top-k famous places* T_kFP query, which enriches the semantics of the conventional spatial query by introducing a social relevance component. We propose three approaches namely, (1) Social-First (2) Spatial-First and (3) Hybrid to efficiently process such queries. In addition, an *FCR-Tree* index structure is proposed that integrates social information with spatial information. Results of empirical studies with an implementation demonstrate the effectiveness of the proposed approaches using real and synthetic datasets.

In the future, it will be interesting to explore the implications of T_kFP query, i.e., aiming to retrieve a result that is as beneficial as possible for an important use case based on dependable ground truth by collecting real data including query workloads.

References

1. Curtiss, M., Becker, I., Bosman, T., Doroshenko, S., Grijincu, L., Jackson, T., Kunnatur, S., Lassen, S., Pronin, P., Sankar, S., Shen, G., Woss, G., Yang, C., Zhang, N.: Unicorn: a system for searching the social graph. PVLDB **6**(11), 1150–1161 (2013)
2. Ahuja, R., Armenatzoglou, N., Papadias, D., Fakas, G.J.: Geo-social keyword search. In: Claramunt, C., Schneider, M., Wong, R.C.-W., Xiong, L., Loh, W.-K., Shahabi, C., Li, K.-J. (eds.) SSTD 2015. LNCS, vol. 9239, pp. 431–450. Springer, Heidelberg (2015)

3. Armenatzoglou, N., Ahuja, R., Papadias, D.: Geo-social ranking: functions and query processing. *VLDB J.* **24**(6), 783–799 (2015)
4. Emrich, T., Franzke, M., Mamoulis, N., Renz, M., Züfle, A.: Geo-social skyline queries. In: Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B. (eds.) *DASFAA 2014, Part II*. LNCS, vol. 8422, pp. 77–91. Springer, Heidelberg (2014)
5. Doytsher, Y., Galon, B., Kanza, Y.: Managing socio-spatial data as large graphs. In: *WWW* (2012)
6. Liu, W., Sun, W., Chen, C., Huang, Y., Jing, Y., Chen, K.: Circle of friend query in geo-social networks. In: Lee, S., Peng, Z., Zhou, X., Moon, Y.-S., Unland, R., Yoo, J. (eds.) *DASFAA 2012, Part II*. LNCS, vol. 7239, pp. 126–137. Springer, Heidelberg (2012)
7. Yang, D.N., Shen, C.Y., Lee, W.C., Chen, M.S.: On socio-spatial group query for location-based social networks. In: *KDD 2012*, Beijing (2012)
8. Armenatzoglou, N., Papadopoulos, S., Papadias, D.: A general framework for geo-social query processing. In: *Proceedings of the VLDB Endowment* (2013)
9. Ference, G., Ye, M., Lee, W.C.: Location recommendation for out-of-town users in location-based social networks. In: *22nd ACM, CIKM*, San Francisco (2013)
10. Mouratidis, K., Li, J., Tang, Y., Mamoulis, N.: Joint search by social and spatial proximity. *IEEE Trans. Knowl. Data Eng.* **27**(3), 781–793 (2015)
11. Doytsher, Y., Galon, B., Kanza, Y.: Querying geo-social data by bridging spatial networks and social networks. In: *LBSN*, San Jose (2010)
12. Huang, Q., Liu, Y.: On geo-social network services. In: *17th International Conference on Geoinformatics*, 2009, pp. 1–6. IEEE (2009)
13. Ye, M., Yin, P., Lee, W.C.: Location recommendation for location-based social networks. In: *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 458–461. ACM (2010)
14. Sarwat, M., Levandoski, J.J., Eldawy, A., Mokbel, M.F.: Lars*: an efficient and scalable location-aware recommender system. *IEEE Trans. Knowl. Data Eng.* **26**, 1384–1399 (2014)
15. Gao, H., Liu, H.: Data analysis on location-based social networks. In: Chin, A., Zhang, D. (eds.) *Mobile Social Networking*, pp. 165–194. Springer, New York (2014)
16. Li, J., Cardie, C.: Timeline generation: tracking individuals on twitter. In: *23rd International World Wide Web Conference, WWW 2014*, Seoul (2014)
17. Li, G., Chen, S., Feng, J., Tan, K. L., Li, W.S.: Efficient location-aware influence maximization. In: *SIGMOD, Snowbird* (2014)
18. Wu, D., Li, Y., Choi, B., Xu, J.: Social-aware top-k spatial keyword search. In: *IEEE MDM*, 2014, Brisbane (2014)
19. Jiang, J., Lu, H., Yang, B., Cui, B.: Finding top-k local users in geo-tagged social media data. In: *31st IEEE ICDE 2015*, Seoul (2015)
20. Memcached. <http://memcached.org/>
21. Twitter: Real-time Geo. <http://slideshare.net/raffikrikorian/rtgeo-where-20-2011>
22. GeoSpatial indexes in MongoDB. <http://docs.mongodb.org/manual/core/geospatial-indexes/>
23. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: *SIGMOD 1984*, pp. 47–57 (1984)
24. Cho, E., Myers, S.A., Leskovec, J.: Friendship, mobility: user movement in location-based social networks. In: *KDD*. ACM (2011)

Databases Theory and Applications

27th Australasian Database Conference, ADC 2016,

Sydney, NSW, September 28-29, 2016, Proceedings

Cheema, M.A.; Zhang, W.; Chang, L. (Eds.)

2016, XXXVI, 486 p. 181 illus., Softcover

ISBN: 978-3-319-46921-8