Location-Aware Group Preference Queries in Social-Networks

Ammar Sohail, Arif Hidayat, Muhammad Aamir Cheema, and David Taniar

Faculty of Information Technology, Monash University, Australia
{ammar.sohail, arif.hidayat, aamir.cheema,
david.taniar}@monash.edu

Abstract. With the recent advances in location-acquisition techniques and *GPS*-embedded mobile devices, traditional social networks such as Twitter and Facebook have acquired the dimension of location. This in result has facilitated the generation of geo-tagged data (e.g., check-ins) at unprecedented scale and have essentially enhanced the user experience in location-based services associated with social networks. Typical location-based social networks allow people to `check-in` at a location of interest using smart devices which then is published on social network and this information can be exploited for recommendation. In this paper, we propose a new type of query called *Geo-Social Group preference Top-k* (*SG-*Top_k) query. For a group of users, a *SG-*Top_k query returns top-k places that are most likely to satisfy the needs of users based on spatial and social relevance. Finally, we conduct an exhaustive evaluation of proposed schemes to answer the query and demonstrate the effectiveness of the proposed approaches.

1 Introduction

A location-based social network (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 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 [1]. In addition, the nodes and edges may also contain spatial information such as a user's check-ins at different locations [2]. Consider the example of a Facebook user Sarah who was born in USA, works at Monash University and checks-in at a particular restaurant. Facebook records this information by linking Facebook pages for *Monash University* and *USA* with Sarah [3], e.g., Sarah and Monash University are connected by an edge labelled `works-at` and, Sarah and USA are connected with an edge labelled `born-in`. The check-in information records the places the user has visited.

Inarguably, social connections play a vital role in our daily lives to enable us in making right decisions in various activities and events and thus impose some influence on us. For instance, previous work [4] explores the effects of social influence on recommendation and their experiments show that a user adopts a suggestion from people socially connected to her which may or may not be derived from her own preference. In recent years, several works on social network analysis [5] have observed that a user's behaviour indeed often correlates to the behaviour of her friends.

In many applications, a group of users may want to plan an activity to find a point of interest (POI) for example, some conference attendees would like to go out for dinner together. For this purpose, we may consider their respective locations and social circles to recommend required POIs. In this paper, we study a problem of finding top-k places considering their distance from the group of query users Q and popularity of the place among each query user $q_i \in Q$'s social connections (e.g., the number of check-ins at the place by each q's friends).

Consider an example of a group of tourists visiting Melbourne. The group consists of tourists from various countries e.g., conference attendees from Italy, Germany and Denmark. They may want to find a nearby pub which is popular (e.g., frequently visited) among people from their respective countries. This involves utilizing spatial information (i.e., near by pub, check-ins) as well as social information (i.e., people who were `born-in` Italy, Germany and Denmark).

The applications of such queries are not only limited to traditional location-based social services. These can also be used in disaster management, public health, security, tourism, marketing etc.. For example, in public safety and crime prevention, law enforcement agencies may be keen on finding frequently visited places by users who have tweeted about *Drugs, Burglary and Extortion* and have also joined some pages/groups containing/sharing information related to those crimes on social networks. The users are socially connected through an edge (entity) e.g., a tweet, a page or a group in social network and then agencies can exploit one-hop neighbours of the entities to find frequently visited places to raid and prevent drugs dissemination.

Although several types of queries have been investigated on LBSNs [6–8], to the best of our knowledge, none of the existing methods can be applied to answer the queries like the above that aim at finding near by places that are popular in social circles of the query users satisfying social and spatial constraints. Motivated by this, in this paper, we formalize this problem as a *Geo-Social Group preference Top-k* (SG-Top_k) query and propose efficient query processing techniques. Specifically, a SG-Top_k query retrieves top-k places (points of interest) ranked according to their spatial and social relevance to the group of query users where the spatial relevance is based on how close the place is to the location of each group member and the social relevance is based on how frequently it is visited by the one-hop neighbors of each query user $q_i \in Q$. A formal definition is provided in Section 3.1.

First we present *Branch-and-Bound* approach to solve our problem and then we propose some optimization techniques to further improve its performance. Our experimental study shows that our optimized algorithm outperforms the other one.
We make the following contributions in this paper.
1. To the best of our knowledge, we are the first to study the SG-Top_k query that retrieves near by places popular among a particular group of users w.r.t. each query user $q_i \in Q$ in the social network.
2. We present *Branch-and-Bound* algorithm followed by some optimizations made to it to process the query which enable flexible data management and algorithmic design.
3. We conduct an exhaustive evaluation of the proposed schemes using real dataset and demonstrate the effectiveness of the proposed approaches.

2 Related Work

Geo-Social query processing is an emerging field and is getting attention of research community these days [7, 9]. Huang et al [10] 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. Yang et al [11] introduced a group query 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 [12, 13].

Similarly, top-k queries retrieve top-k objects based on a user defined scoring function and have been well studied [1, 14]. Ilyas et al. [15] give a comprehensive survey of top-k query processing techniques. [16] propose some of the top-k processing algorithms for example Fagins Algorithm and No-Random Access algorithms. Another work is presented by Jiang et al [17] in which they propose a method to find *top-k* local users in geo-social media data. However, their social scoring criteria is not applicable to our problem definition. There are some related works on group queries and community search but they focus on different scenarios and objectives [18, 19].

Group (aggregate) nearest neighbour queries are first presented by Papadias et al. [20] and proposed three different methods MQM (multi query method), SPM (single point method) and MBM (minimum bounding method). The propsed methods are designed for Eucledean space and are not suitable for criteria involving users' preference. In 2007, Yiu et al. [21] introduces a new query called *top-k* spatial preference query which returns top-k objects whose ranking is defined by other objects around them. Yuan et al. [6] proposed a new query which returns top-k objects based on distance and objects ratings. Many previous studies [21, 22] have proposed to integrate POI properties into POI recommendations however, these works do not consider user preferences.

3 Preliminaries

3.1 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 is indicated by a labelled edge where the label indicates the type of relationship (e.g., `friend`, `lives-in`) [1]. A 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 at a particular time.

Score of a place *p* [1, 2]: Given a query user q_i, the score of a place $p \in P$ is the weighted sum of its spatial score (denoted as $spatial(p, q_i)$) and its social score (denoted as $social(p, q_i)$).

$$Score(p, q_i) = \alpha \times spatial(p, q_i) + (1 - \alpha) \times social(p, q_i) \tag{1}$$

where α is a parameter used to control the relative importance of spatial and social scores. The social score $social(p, q_i)$ is computed as follows. Let F_{qi} denotes the one-hop neighbors of any of the query users $q_i \in Q$ considering a particular relationship type, e.g., if the relationship is `works-at`, the query entity is a Facebook Page for the company Samsung, then F_{qi} is a set of users who work in Samsung. 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_{qi} contains the friends of the query user q_i. Let V_p denotes the set of all users that visited (i.e., checked-in at) the place. The social score $social(p, q_i)$ of place p is computed as follows:

$$social(p, q_i) \ = 1 - \frac{|F_{qi} \cap V_p|}{|F_{qi}|} \tag{2}$$

where $|X|$ denotes the cardinality of a set X. Intuitively, $social(p, q_i)$ is the proportion of the friends of a query user $q \in Q$ who have visited the place p.

The spatial score $spatial(p, q_i)$ is based on how close the place is to the query user $q_i \in Q$. Formally, $spatial(p, q_i) = ||p, q_i||$, where $||p, q_i||$ indicates Euclidean distance between the query user and p. Note that $social(p, q_i)$ is always between 0 to 1 and the smaller social score is considered better. In addition, we also normalize $spatial(p, q_i)$ such that it is also between 0 to 1, e.g., the data space is normalized such that $||p, q_i|| \leq 1$.

Aggregate Score of a place *p*: Given a set of query users $Q = \{q_1, q_2, ...q_n\}$, the aggregate score of a place p (denoted as $aggScore(p)$) is computed using a monotonic scoring function f which takes as input each $Score(p, q_i)$ for every $q_i \in Q$.

$$aggScore(p) = f(Score(p, q_i), ..., Score(p, q_n)) \tag{3}$$

For example, if f is *average*, the aggregate score corresponds to $\sum_{i=1}^{i=n} Score(p, q_i)/n$. Similarly, if f is *min*, the aggregate score corresponds to minimum of $Score(p, q_i)$ for $q_i \in Q$.

Geo-Social Group preference Top-k ($SG\text{-}Top_k$) Query: Given a set of places $P = \{p_1, p_2, ...p_n\}$ in a LBSN, a $SG\text{-}Top_k$ query Q returns k places with smallest aggregate score $aggScore(p)$. The $aggScore(p)$ of each place $p \in P$ is computed as described above depending on the function f used. Some examples of the function f are *min, max and avg*. For instance, if f corresponds to min, the aggregate score will be $aggScore(p) = min(Score(p, q_i), ..., Score(p, q_n))$ that is, $aggScore(p)$ = $\arg\min_{i:1\ to\ n} Score(p, q_i)\ \forall\ q_i \in Q$. As an example, consider a dataset containing a set of places i.e., $P = \{p_1, p_2, p_3, p_4, p_5\}$ and Q is a set of query users i.e., $Q = \{q_1, q_2, q_3\}$. Table 1 illustrates spatial score, social score, and score for each place p for each query user q_i and it also shows the aggregate score.

P	$\|\|p, q_i\|\|$	$Social(p, q_i)$	$Score(p, q_i)$	$aggScore(p)$
p_1	$q_1 = 0.10$	$q_1 = 0.6$	$q_1 = 0.35$	$Avg = 0.36$
	$q_2 = 0.14$	$q_2 = 0.4$	$q_2 = 0.27$	$min = 0.27$
	$q_3 = 0.12$	$q_3 = 0.8$	$q_3 = 0.46$	$max = 0.46$
p_2	$q_1 = 0.09$	$q_1 = 0.8$	$q_1 = 0.445$	$Avg = 0.345$
	$q_2 = 0.05$	$q_2 = 0.6$	$q_2 = 0.325$	$min = 0.265$
	$q_3 = 0.13$	$q_3 = 0.4$	$q_3 = 0.265$	$max = 0.445$
p_3	$q_1 = 0.22$	$q_1 = 1.0$	$q_1 = 0.61$	$Avg = 0.495$
	$q_2 = 0.20$	$q_2 = 0.6$	$q_2 = 0.40$	$min = 0.40$
	$q_3 = 0.15$	$q_3 = 0.8$	$q_3 = 0.475$	$max = 0.61$
p_4	$q_1 = 0.20$	$q_1 = 0.2$	$q_1 = 0.20$	$Avg = 0.245$
	$q_2 = 0.17$	$q_2 = 0.4$	$q_2 = 0.285$	$min = 0.20$
	$q_3 = 0.10$	$q_3 = 0.4$	$q_3 = 0.25$	$max = 0.285$
p_5	$q_1 = 0.17$	$q_1 = 0.6$	$q_1 = 0.385$	$Avg = 0.363$
	$q_2 = 0.12$	$q_2 = 0.8$	$q_2 = 0.46$	$min = 0.245$
	$q_3 = 0.09$	$q_3 = 0.4$	$q_3 = 0.245$	$max = 0.46$

Table 1: Sample Dataset and Aggregate Scores

If f corresponds to avg, $\alpha = 0.5$ and $k = 2$, the corresponding $SG\text{-}Top_2$ query reports places (p_4 and p_2) that minimizes the average aggregate score $aggScore(p)$. Similarly, if f corresponds to max, the $SG\text{-}Top_2$ query reports places (p_4 and p_2) that minimizes the maximum aggregate score ($aggScore(p)$). On the other hand, if f corresponds to min, the $SG\text{-}Top_2$ query reports places (p_4 and p_5) that minimizes the minimum aggregate score ($aggScore(p)$).

4 Techniques Overview

Before presenting our approaches to solve $SG\text{-}Top_k$ query in detail, first we provide a brief overview of the approaches and to the best of our knowledge, there does not exist any technique in literature that can be adopted to answer the proposed query. First we present *Branch-and-Bound* approach and then we present optimization techniques to further improve its performance. The *Branch-and-Bound* approach uses R-tree to process places in ascending order of their aggregate distance from each query user $q_i \in Q$ and then computes social score of each place p to finally compute aggregate score i.e., $aggScore(p)$. Since computation of $aggScore(p)$ of each $p \in P$ is computationally expensive, we design a specialized index structure associated with each user $u \in U$ to pre-process her friends' check-in information. Our Optimized approach leverages the proposed index structure to offer efficient pruning techniques to prune such candidate places that cannot be a part of top-k places.
Next, we briefly describe three indexes used by our algorithms.
Facility R-Tree: We create an R-tree where all places ($p \in P$) in the dataset are indexed based on their location coordinates.
Check-in R-Tree: For each user u, we create a *Check-In R-Tree* which indexes all check-ins of the u. This is a 2 dimensional R-tree containing the location coordinates information of each check-in. If a place p is visited by a user multiple times, it will be indexed as many times it was visited hence, *Check-in R-Tree* contains duplicate entries for the place p since many applications (e.g., which include ranking and recommendation of places) do require complete check-in information of users.
Friendship Index: For each user, her friends are indexed using B^+*-Tree* sorted on their IDs. This is used to efficiently retrieve the friends based on their IDs.

4.1 Branch-and-Bound (B&B) Algorithm

Before presenting our *Optimized* approach, we first discuss *B&B* approach to process $SG\text{-}Top_k$ query. The B&B approach is to traverse *Facility R-Tree* in *best-first* manner. For this purpose, we use a $min\text{-}heap$ where the key for each entry E is $minAggDist(Q, E)$. To compute $minAggDist(Q, E)$, the algorithm computes minimum distance of E from each query user $q_i \in Q$ and then according to the aggregate function f provided, the algorithm finalises the value of $minAggDist(Q, E)$. For example, if $f = min$, the $minAggDist(Q, E)$ is the smallest minimum distance between E and any of the q_i. Then, we initialize $min\text{-}heap$ with the root of *Facility R-Tree*.

Further, the algorithm starts de-heaping entries in ascending $minAggDist(Q, E)$ order. If a de-heaped entry E is a node, it computes its lower-bound aggregate score (denoted as $LBaggScore(E)$) based on its $minAggDist(Q, E)$ only, assuming that its social score i.e., $social(E)$ is maximum. Further, if the de-heaped entry E is an object (a place p), it computes its exact aggregate score $aggScore(p)$ and updates *top-k* places. Finally, at any point, if an entry E having lower-bound aggregate score worse than current k_{th} best place score (denoted as $aggScore_k$) is de-heaped, the algorithm terminates. The reason is, every subsequent entry E in $min\text{-}heap$ will have worse aggregate score than the current $aggScore_k$.

4.2 Optimized Algorithm

This section focuses on our *Optimized* approach to process $SG\text{-}Top_k$ queries and before presenting the technique in detail, first we describe a specialized index specifically designed for the technique.

Friends Check-ins R-Tree: In addition to the previous indexes, for each user u, we propose another index called *Friends Check-Ins R-tree (FCR-Tree)* which maintains the *summary* of check-ins of all friends of u. Specifically, *FCR-Tree* stores check-in information of each friend of u by indexing only one MBR for each friend, thus constitutes the summary of all friends check-ins. Since we index only one object per friend therefore, the size of the index is proportional to the number of friends of u. Note that the indexed objects are the root MBRs of each friend's *Check-in R-Tree*.

Let's assume a user $u \in U$ where the friends of u are $F_u = \{u_1, u_2, u_3 \ldots u_{19}, u_{20}\}$. Figure 1 illustrates the idea behind the *FCR-Tree*.

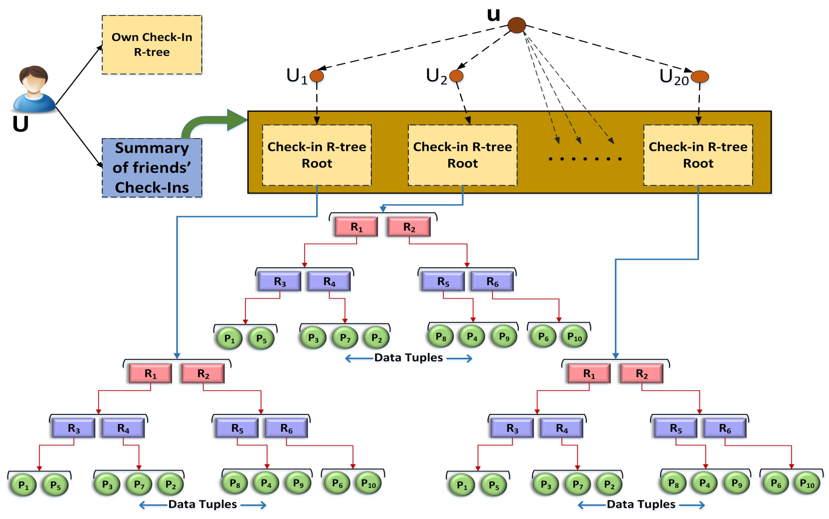


Fig. 1: Summary of Friends' check-ins

4.2.1 Computation Module:

In B&B algorithm, since lower-bound of *Facility R-tree* nodes is computed based only on $minAggDist(Q, E)$, it is loose which results in high computation cost. To develop an efficient approach to computing solution for $SG\text{-}Top_k$ query, we propose few improvements in the algorithm. First, we present highlights of the improvements we made.

1. First we compute tighter lower-bound on aggregate score of a node entry E i.e., $LBaggScore(E)$ using better estimate of its social score. For this purpose, the algorithm exploits *FCR-Tree* of each query user $q_i \in Q$.
2. Then, the algorithm computes a region (denoted as region bounded rectangle, *RBR*) based on current $aggScore_k$ to quickly check the entries that can be pruned.
3. Finally, while en-heaping an entry E, we make an observation to avoid computing its lower-bound on social score by associating its parent node's lower-bound on social score with it as an initial estimate.

The details of the above mentioned improvements are provided next.

1. Computing Lower bound on Aggregate score:

Recall that in B&B approach, to estimate best possible aggregate score $LBaggScore(E)$ of an entry E, we assume that its social score i.e., $social(E)$ is maximum and therefore, its lower-bound is loose. To overcome this limitation, the *Optimized* algorithm leverages *Friends Check-ins R-Tree* (FCR-Tree) to estimate social score of the E (denoted as $LBSocial(E)$) which in return, tightens the lower-bound on aggregate score $LBaggScore(E)$.

Specifically, to compute $LBSocial(E)$ of an entry E of *Facility R-tree*, the algorithm traverses specialized index i.e., *FCR-Tree* of a query user q_i to compute number of its objects (root MBRs of Check-In R-trees of friends) intersecting with E. Let's consider an example in Figure 2 where we have a *Facility R-tree* entry E and some *FCR-Tree* objects belonging to q_i's friends ranging from u_1 *to* u_5. Since only u_1, u_4 *and* u_5 overlap with E, they might have checked-in at any place p in E. Therefore, the maximum number of friends who might have visited a place in E is 3, which can be used to obtain the lower-bound on social score. Let's denote the number of overlapping objects as $numOverlap$, the lower-bound on social score is computes as $1 - \frac{numOverlap}{|F_{qi}|}$.

In addition, once the lower-bound on social score against a query user q_i is computed, the lower-bound on score (denoted as $LBScore(E, q_i)$) is computed against the query user q_i using equation 4. The pseudocode of computing $LBScore(E, q_i)$ is given in Algorithm 1.

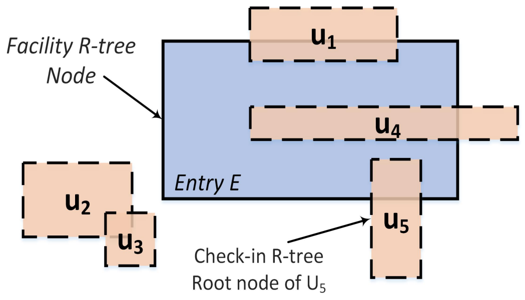


Fig. 2: MBR Social Score Bound

$$LBScore(E, q_i) = \alpha \times minDist(E, q_i) + (1 - \alpha) \times \left(1 - \frac{numOverlap}{|F_{qi}|}\right) \quad (4)$$

Algorithm 1: Get-$LBaggScore(mbr, Q)$

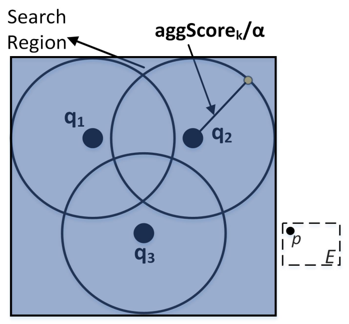
```
1 $numOverlap = \emptyset$;
2 foreach user $q_i \in Q$ do
3     Issue a range query on FCR-Tree;
4     $numOverlap$ = Compute number of objects overlapping with the $mbr$;
5     $LBSocial(mbr, q_i) = 1 - \frac{numOverlap}{|F_{qi}|}$;
6     $LBScore(mbr, q_i) = \alpha \times minDist(mbr, q_i) + (1 - \alpha) \times LBSocial(mbr, q_i)$ ;
7 end
8 $LBaggScore(mbr) = f(LBScore(mbr, q_i), ..., LBScore(mbr, q_n))$;
9 return $LBaggScore(mbr)$
```

2. Optimal MBR based Search Regions:
Recall that in B&B, we need to compute $minDist(E, q_i)$ n times to compute $minDist(E, Q)$. For this purpose, we require to compute minimum distance from the entry E to each q_i and if this distance is greater than $\frac{aggScore_k}{\alpha}$, we can ignore E. However, this requires

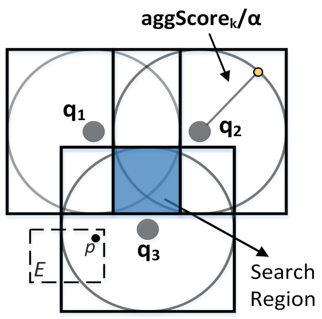
computing $minDist(E, q_i)$ n times. To overcome this problem, we create a *Region Bounding Rectangle* (denoted as *RBR*) such that if an entry E does not overlap with it, it is pruned.

In particular, *RBR* is defined by corresponding aggregate function f and its size depends on current $aggScore_k$ and α i.e., $\frac{aggScore_k}{\alpha}$. The algorithm only accesses those entries which intersect with it. Figure 3 demonstrates two *RBR*s for *min, max and average* aggregate functions.

- ***Min***: Consider $SG\text{-}Top_k$ query with $Q = \{q_1, q_2, q_3\}$ in Figure 3(a). The shaded area corresponds to the RBR for *min* (where n = 3) which is a minimum bounding rectangle of union of three circles (centred at q_1, q_2, q_3) each with radius $\frac{aggScore_k}{\alpha}$. Entry E for example, should be not visited since it is not intersecting with the RBR and cannot contain a place p whose smallest score w.r.t. any of the $q_i \in Q$ is smaller than current $aggScore_k$ i.e., for any place p in E, $aggScore(p) > aggScore_k$.
- ***Max***: The RBR corresponds to the intersection of three minimum bounding rectangles (shaded area) for each circle (centred at corresponding q_i) with radius $\frac{aggScore_k}{\alpha}$ as illustrated in Figure 3(b). Let's take an example of an entry E intersecting with the circle of q_3. This entry E should not be visited because for any place p in E, which is outside this intersected area, $dist(q_i, p) > aggScore_k$ for at least one q_i. Therefore, $aggScore(p) > aggScore_k$.
- ***Average***: For *f=Average*, the RBR is same as *f=min* because a place p for which $aggScore(p)$ regarding *f=min* is greater than $aggScore_k$, its $aggScore(p)$ regarding *f=Average* is also greater than $aggScore_k$. Note that a place p that is outside RBR, has $aggScore(p) > aggScore_k$ because its average distance is greater than $\frac{aggScore_k}{\alpha}$. For example, an entry E should not be visited since it is not intersecting with the RBR as shown in Figure 3(a). Therefore, it cannot contain a place p whose average score (aggScore(p)) w.r.t. all $q_i \in Q$ is smaller than current $aggScore_k$.



(a) Search Region (*f=avg, min*)



(b) Search Region (*f=max*)

Fig. 3: MBR Based Search Space

3. Observation on Social Score:
Recall that in step 1, to compute lower-bound aggregate score of an entry E, it requires traversing *FCR-Tree* another time while inserting it in priority queue for the first time. To avoid this overhead, we consider its parent's social score lower-bound as an initial estimate of $E's$ social score lower-bound which is a valid lower-bound on a child's social score.

We next describe the algorithm in detail with pseudocode given in Algorithm 2.

4.2.2 Algorithm Overview:

Algorithm 2 starts traversing *Facility R-tree* in best-first approach. For this purpose, a *min-heap* is initialized with the root node with $LBaggScore(root)$ as a sorting key (at line 3). To compute $LBaggScore$, it first invokes *Get-LBaggScore(mbr,Q)* (algorithm 1). Then in first loop (at line 4), algorithm starts de-heaping entries iteratively and examines whether or not it intersects with *RBR*, if it does not, it is immediately pruned along with all the entries lie inside (at line 6). Further, if the entry E overlaps and is a place p (an object), the algorithm computes its $aggScore(P)$ (at line 8) and update current k_{th} best place score $aggScore_k$ and *RBR* (at line 9).

Algorithm 2: Optimized Algorithm

```
1  $min\text{-}heap = \emptyset, aggScore_k = \infty$;
2  $LBaggScore$ = Get-$LBaggScore(root, Q)$;
3  Initialize min-heap with root of Facility R-tree with $LBaggScore$ as a key ;
4  while $min\text{-}heap \neq \emptyset$ do
5      De-heap entry $E$;
6      if $E$ overlaps with $RBR$ then
7          if $E$ is an object then
8              Compute $aggScore(E)$;
9              Update $aggScore_k$ and $RBR$;
10         else
11             $LBaggScore(E)$ = Get-$LBaggScore(E, Q)$ // Algorithm 1
12             if $LBaggScore(E) < aggScore_k$ then
13                 foreach child node c of E do
14                     if c overlaps with $RBR$ then
15                         $LBaggScore(c) =$
                           $\alpha \times minAggDist(c, Q) + (1 - \alpha) \times LBsocial(E)$;
16                         if $LBaggScore(c) < aggScore_k$ then
17                             insert c in $min\text{-}heap$;
18                 end
19 end
20 return Return Top-k Places
```

Otherwise, the algorithm invokes *Get-LBaggScore(mbr,Q)* (algorithm 1) to compute $LBaggScore(E)$ (at line 11). Subsequently, if $LBaggScore(E)$ is better than current $aggScore_k$, in second loop, it starts en-heaping its child nodes provided that they overlap with *RBR* (at line 14). Moreover, if a child node c qualifies, the algorithm computes its $LBaggScore(c)$ by inheriting its social score from its parent. Consequently, if the estimated $LBaggScore(c)$ of c is less than the current $aggScore_k$, it is finally en-heaped for further processing (at line 17). Once $min\text{-}heap$ is emptied, the algorithm terminates and reports top-k places (at line 20).

5 Experiments

5.1 Experimental Setup

To the best of our knowledge, this problem has not been studied before and no previous algorithm can be trivially extended to answer $SG\text{-}Top_k$ queries therefore, we evaluate the proposed algorithms on their performance by comparing them with each other.

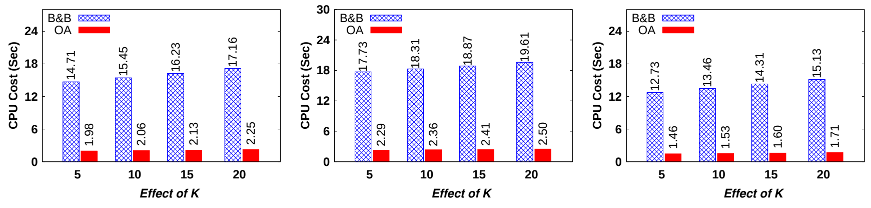
Each method is implemented in C++ and experiments are run on Intel Core *I*5 2.4GHz PC with 16GB memory running on 64-bit Ubuntu Linux. We use real dataset of *Gowalla* [23] and various parameters used in our experiments are shown in Table 2 where *Query MBR Size* represents the size of the region in which query users are spread. *Gowalla* dataset contains 196,591 users, 950,327 friendships, 6,442,890 check-ins and 1,280,956 checked-in places across the world. The node size of *Facility R-tree* index is set to 4096 Bytes and 1024 Bytes for *Check-In R-Tree* and *FCR-Tree* indexes because they have fewer objects as compared to *Facility R-tree*. For each experiment, we randomly choose 10 groups of query users and consider them as query groups Q.

Parameters	**Values**
Group Size (n)	2, **4**, 6, 8
f	*min*, *max*, *avg*
Query MBR Size (km)	50, **100**, 200, 400
Average Friends	200, 400, **600**, 800
k	5, **10**, 15, 20

Table 2: Parameters (Default shown in bold)

5.2 Performance Evaluation

Effect of k: In this evaluation, we test our proposed algorithms for various values of k for *min, max and avg* function. Note that for *f* = *min* in Figure 4(a), *Optimized Algorithm* (OA) is upto 10 times faster and the performance is not significantly affected by the value of k. The reason is that, the main cost depends on traversing *Facility R-tree* and then computing lower bounds of the nodes and this dominant cost is not affected by k. Similarly, in Figure 4(b) for *f* = *max*, *Branch-and-Bound Algorithm* (B&B) takes little bit longer to process the query due to more number of places being qualified as candidates. However, for *f* = *avg* both the algorithms performs better as compared to other functions where *OA* outperforms *B&B* by atleast 8 time as shown in Figure 4(c).

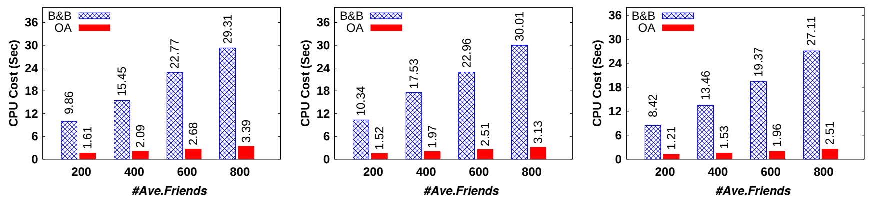


(a) f=min (b) f=max (c) f=avg

Fig. 4: Effect of varying number of requested places (k)

Effect of Average number of Friends: In this experiment, we study the effect of number of friends on *B&B* and *OA* algorithms in Figure 5. Note that the size of *FCR-Tree* relies on number of friends of a query user $q_i \in Q$. Also, the distribution of each friend's check-ins in search space determines the size of root node of *Check-in R-Tree*. This in return, affects the lower-bound on social score of *Facility R-tree* entries in *OA* Algorithm. In *B&B* algorithm, CPU cost mainly depends on computing the social score of the places $p \in P$ and as we increase the number of friends, the CPU cost increases. On the other hand, *OA* algorithm is less affected due to the optimization techniques. Note that, if f=avg, both the algorithms perform better than *min and max* functions due to lower $aggScore_k$ which aids in pruning more places.

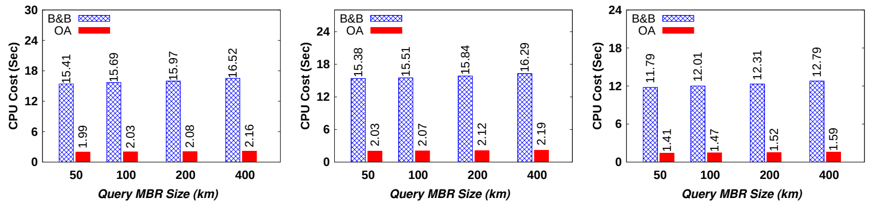
Effect of Query MBR Size: Next in Figure 6, we evaluate the performance of our algorithms on query MBR size. For this purpose, we randomly spread the query users in the region of size between 50 to 400 kilometres. Note that, as we increase the size,



(a) f=min (b) f=max (c) f=avg

Fig. 5: Effect of varying number of Friends

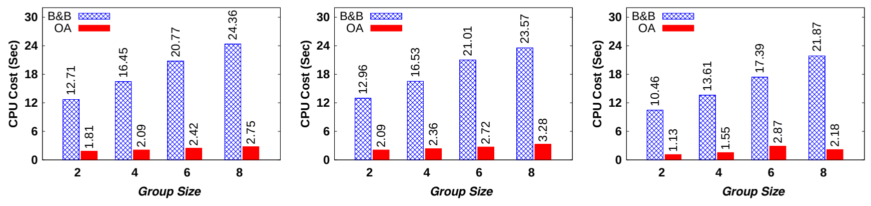
it does not affects the query processing to great extent since the main cost involves traversing *Facility R-tree*, computing lower-bounds and social score of the query. Note that for *f* = *min* and *f* = *max* in Figure 6(a) and 6(b) respectively, *Optimized Algorithm* (OA) is upto 10 times faster *B&B* algorithm. However, if *f* = *average*, the algorithms perform relatively better as illustrated in Figure 6(c).



(a) f=min (b) f=max (c) f=avg

Fig. 6: Effect of varying Query MBR Size

Effect of Group Size: In this evaluation, we test our proposed algorithms for different group sizes ranging from 2 to 8 for *min, max and avg* aggregate functions in Figure 7. As we increase the group size, it greatly affects the performance of the two algorithms because the algorithms have to process spatial and social information for more query users. However, for larger groups, *OA* performs better than the other one. In addition, if *f* = *avg*, both the algorithms perform relatively better than *min and max* as shown in Figure 7(c).



(a) f=min (b) f=max (c) f=avg

Fig. 7: Effect of varying Group Size (n)

6 Conclusions

In this paper, we formalized a problem namely, *Geo-Social Group preference Top-k* ($SG\text{-}Top_k$) query and proposed efficient query processing techniques. First we presented *Branch-and-Bound* approach to solve our problem and then we proposed some optimization techniques to further improve its performance. For this purpose, we introduced a specialized index structure , a region bounding rectangle and an observation to efficiently process the query. Our experimental study showed that our optimized algorithm outperforms the other one.

References

1. Ammar Sohail, Ghulam Murtaza, and David Taniar. Retrieving top-k famous places in location-based social networks. In *ADC*, 2016.
2. Ammar Sohail, Muhammad Aamir Cheema, and David Taniar. Social-aware spatial top-k and skyline queries. *The Computer Journal*, 62, 2018.
3. Michael Curtiss et al. Unicorn: A system for searching the social graph. *PVLDB*, 2013.
4. Mao Ye, Xingjie Liu, and Wang-Chien Lee. Exploring social influence for recommendation: a generative model approach. In *SIGIR*, 2012.
5. Timothy La Fond and Jennifer Neville. Randomization tests for distinguishing social influence and homophily effects. In *WWW*, 2010.
6. Yuan Tian, Peiquan Jin, Shouhong Wan, and Lihua Yue. Group preference queries for location-based social networks. In *APWeb-WAIM*, 2017.
7. Nikos Armenatzoglou, Ritesh Ahuja, and Dimitris Papadias. Geo-social ranking: functions and query processing. *VLDB J.*, 2015.
8. Yuqiu Qian, Ziyu Lu, Nikos Mamoulis, and David W. Cheung. P-LAG: location-aware group recommendation for passive users. In *SSTD*, 2017.
9. Kyriakos Mouratidis, Jing Li, Yu Tang, and Nikos Mamoulis. Joint search by social and spatial proximity. *IEEE Trans. Knowl. Data Eng.*, 27(3):781–793, 2015.
10. Qian Huang and Yu Liu. On geo-social network services. In *Geoinformatics, 2009 17th International Conference*, pages 1–6. Ieee, IEEE, New York, NY, USA, 2009.
11. De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. On socio-spatial group query for location-based social networks. In *KDD*, 2012.
12. Mohamed Sarwat, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. Lars*:an efficient and scalable location-aware recommender system. *Knowl. Data Eng.*, 2014.
13. Huiji Gao and Huan Liu. Data analysis on location-based social networks. In *Mobile social networking*, pages 165–194. Springer, Berlin Heidelberg, 2014.
14. Dingming Wu, Yafei Li, Byron Choi, and Jianliang Xu. Social-aware top-k spatial keyword search. In *MDM*, 2014.
15. Ihab F. Ilyas, George Beskales, and Mohamed A. Soliman. A survey of top-*k* query processing techniques in relational database systems. *ACM Comput. Surv.*, 2008.
16. Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.*, 2003.
17. Jinling Jiang, Hua Lu, Bin Yang, and Bin Cui. Finding top-k local users in geo-tagged social media data. In *ICDE*, 2015.
18. Theodoros Lappas, Kun Liu, and Evimaria Terzi. Finding a team of experts in social networks. In *SIGKDD*, pages 467–476, 2009.
19. Cheng-Te Li and Man-Kwan Shan. Team formation for generalized tasks in expertise social networks. In *IEEE, SocialCom / IEEE, PASSAT*, 2010.
20. Dimitris Papadias, Qiongmao Shen, Yufei Tao, and Kyriakos Mouratidis. Group nearest neighbor queries. In *Data Engineering*. IEEE, 2004.
21. Man Lung Yiu, Xiangyuan Dai, Nikos Mamoulis, and Michail Vaitis. Top-k spatial preference queries. In *ICDE*, 2007.
22. Muhammad Attique, Hyung-Ju Cho, Rize Jin, and Tae-Sun Chung. Top-*k* spatial preference queries in directed road networks. *ISPRS Int. J. Geo-Information*, 2016.
23. Eunjoon Cho, Seth A. Myers, and Jure Leskovec. Friendship and mobility: user movement in location-based social networks. In *ACM SIGKDD*, 2011.