Noname manuscript No.
(will be inserted by the editor)

Geo-Social Ranking: Functions and Query Processing

Nikos Armenatzoglou · Ritesh Ahuja · Dimitris Papadias

Received: date / Accepted: date

Abstract Given a query location q, Geo-Social Ranking (GSR) ranks the users of a Geo-Social Network based on their distance to q, the number of their friends in the vicinity of q, and possibly the connectivity of those friends. We propose a general GSR framework and four GSR functions that assign scores in different ways: i) LC, which is a weighted linear combination of social (i.e., friendships) and spatial (i.e., distance to q) aspects, ii) RC, which is a ratio combination of the two aspects, iii) HGS, which considers the number of friends in coincident circles centered at q, and iv) GST, which takes into account triangles of friends in the vicinity of q. We investigate the behavior of the functions, qualitatively assess their results, and study the effects of their parameters. Moreover, for each ranking function, we design a query processing technique that utilizes its specific characteristics to efficiently retrieve the top-k users. Finally, we experimentally evaluate the performance of the top-k algorithms with real and synthetic datasets.

1 Introduction

Geo-Social Networks (GeoSNs) that capture social relations of users and their locations are increasingly popular. Foursquare supports over 45 million users, who have checked-in more than 5 billion times at over 1.6 million businesses [1]. Moreover, even conventional social networks, such as Facebook and Twitter, have expanded their services by introducing check-in functionality. The combination of spatial and social information has generated novel opportunities for marketing and location-based advertising, and additional requirements for query processing. For instance, Foursquare has cooperated with GroupOn to provide offers from merchants to users nearby [2]. This information, when posted on users' public profiles may influence their friends to visit the same places.

Given a location q, *Geo-Social Ranking* (GSR) ranks the users of a GeoSN based on their distance to q, the number of their friends in the vicinity of q, and possibly the connectivity of those friends. As an example, assume that a merchant wishes to post an advertisement using a location-based service; promising targets are users with high GSR scores since in addition to being nearby merchant's location, they can also influence their friends in the vicinity to visit. As another example, [20] describes a dataset containing the locations and social interactions among street gang members in Los Angeles, as observed by police officers. GSR on this dataset can be used to identify possible suspects at locations with high concentration of connected gang members.

Although GeoSNs have attracted a considerable attention in recent years, currently there is no work on the retrieval of the top users based on their spatial and social characteristics with respect to a query. Previous approaches i) focus on retrieving groups instead of individual users [23], ii) are restricted to users with a specified number of friends [3][14], or iii) take into account only the distances among users, instead of their distances to a specific location [26]. In this paper, we propose several GSR functions, and associated top-k

N. Armenatzoglou ( ) · R. Ahuja · D. Papadias
Hong Kong University of Science and Technology, Department of Computer Science and Engineering
E-mail: nikos@cs.ust.hk

R. Ahuja
E-mail: rahuja@cs.ust.hk

D. Papadias
E-mail: dimitris@cs.ust.hk

query processing techniques, suitable for a wide range of applications with different characteristics.

Figure 1 motivates the need for diverse GSR functions. The grey points refer to the locations of 14 users, the edges represent their social relations, and the black star is a query location q. The table shows the Euclidean distance $||q, v_i||$ between q and each user v_i. In some application, v_1 may be considered the top-1 user because he is the closest to q, and has two friends (v_2, v_4) that are very near q. In another scenario, user v_4 may be the best, despite being farther than v_1, since he could influence 5 friends (v_1, v_3, v_5, v_7, v_8) in the area around q. Finally, v_3 could also be considered the most highly ranked user because he has 3 friends (v_4, v_6, v_7), reasonably close to q, and tightly connected to each other (i.e., the subgraph containing v_3, v_4, v_6, v_7 almost forms a clique).

In order to capture different application requirements, we introduce a general Geo-Social ranking framework and propose four functions that cover several practical scenarios: i) Linear Combination (LC), ii) Ratio Combination (RC), both of which are GSR adaptations of two popular ranking functions used in related domains, e.g., in spatial-keyword search [10][22], iii) h-Geo-Social (HGS) ranking function, inspired by the bibliographic h-index, which assigns each user a score based on the number of friends in coincident circles centered at q, and iv) Geo-Social Triangles (GST) ranking function, which in addition to distance, takes into account the friends that a user and his friends have in common, i.e., triangles that are close to the query point.

LC provides a natural way to express real-life constraints, such as the fact that an advertiser is only interested in users within a range; e.g., a restaurant sending lunch promotions to potential customers within $0.5km$. RC can be used in cases where locality is crucial. For instance, a cinema has empty seats for a film starting soon, and sends coupons to users in close proximity. On the other hand, HGS is useful for cases where the distance aspect is not critical; e.g., a concert promotion targeting users with many friends in the wider area of the concert. GST is suitable for applications where connectivity is essential or desirable for ranking; e.g., a promotion similar to that of the concert, but this time for an event (party) that involves social interaction among the various users.

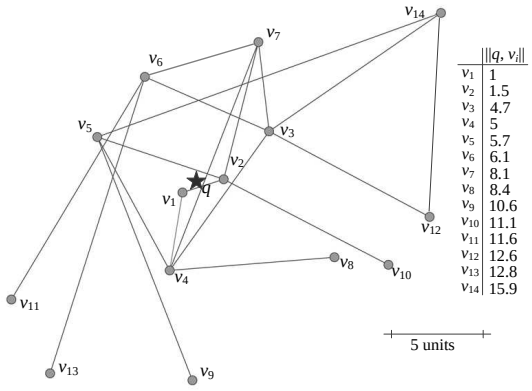


Fig. 1 Running Example

We investigate the behavior of the functions in depth, qualitatively assess their results, examine their correlation, and study the effects of their parameters. Moreover, we design query processing techniques that utilize the individual function characteristics to efficiently retrieve the top-k users. Specifically, we show that some functions can be processed by range queries, while others require incremental retrieval of the results based on the branch and bound framework. All processing algorithms utilize a well defined set of primitive operations that are supported by the majority of commercial GeoSN APIs. Consequently, the proposed techniques can be effortlessly adapted by popular GeoSN, or other novel applications that can access those APIs. The contributions of the paper are summarized below.

- We introduce the GSR problem and propose four diverse functions that rank users based on their social connections and their distances to an input location.
- We develop specialized algorithms that retrieve the top-k users according to each function.
- We visualize the top users of different functions using a real geo-social dataset, examine the effect of their parameters, measure their correlation using Kendall's τ_b rank coefficient [13], and discuss their suitability to different application requirements.
- We experimentally evaluate the performance of the query processing techniques using real and synthetic data.

The rest of the paper is organized as follows. Section 2 overviews related work. Section 3 formalizes the problem of retrieving the top-k users in GeoSNs. Sections 4 to 7 propose the ranking functions and the corresponding query processing methods. Section 8 contains a qualitative evaluation of the ranking functions using a real dataset. Section 9 compares the efficiency of the top-k algorithms experimentally. Finally, Section 10 concludes the paper with directions for future work.

2 Related Work

Given a query point q and two positive integers m, k, the Nearest Star Group query (NSG) returns the k star

groups of m members nearest to q; each group must have a user who is friend with all the other users in the group [3]. Given a query point q and two positive integers m, p, the Socio-Spatial Group query (SSG) returns a group of m users that (i) minimizes the total distance to q, and (ii) each user is on the average connected to at least $m-p$ other group members [23].

The above queries retrieve groups, instead of individual users. The input parameter m on the group size causes undesirable complications for GSR retrieval of single users. For instance, although NSG can rank the central user of a star group, it only returns users with at least $m-1$ friends; moreover, if a user has more than m friends near q, only the $m-1$ closest are considered. The application of SSG to individual users is not obvious. Furthermore, in addition to the group size m, it restricts the social structure (by the $m-p$ constraint).

Given a user v and a positive integer k, the Circle of Friends query (CoF) outputs a subgraph g that consists of v and $k-1$ friends such that both the diameter of g and the maximum distance between any two users in g are minimized [14]. Given a user v, the Social and Spatial Ranking Query ($SSRQ$) finds the top-k users based on their spatial proximity and social connectivity to v [15]. CoF only considers direct friendships, whereas $SSRQ$ also takes into account multi-hop connectivity. Both queries focus on users instead of query points, i.e., they explore the neighborhood of the input user, and are inapplicable to GSR.

Given a user v and a set of users U, the Geo-Social Influence (GSI) metric computes the number of users in U that can be influenced by v based on their social and spatial proximity to v [26]. Given a user v, the Node Locality (NL) metric measures the spatial closeness between v and his friends, while the Geographic Clustering Coefficient (GCC) combines the social clustering coefficient of v (i.e., how close v and his friends are to forming a clique), along with spatial criteria (i.e., each triangle of v has a score based on the spatial proximity of its members) [16]. [17] introduces three more GeoSN metrics: i) Average Distance (AD), ii) Distance Strength (DS), and Average Triangle Length (ATL). AD is the average distance between a user v and his friends. DS is the sum of the distances between v and his friends. The length of a triangle is the sum of the distances between the members of a triangle. ATL of v is the average length of the triangles that v forms with his friends.

All the above metrics are used to identify characteristics of a GeoSN, independent of a query point. Therefore, they are not applicable to GSR. Moreover, they constitute mathematical definitions, without corresponding computation algorithms. Our proposed GST method applies concepts similar to GCC and ATL for GSR. [8] and [18] introduce GeoSN metrics that predict friendships based on the history check-ins. Finally, the GeoSNs queries of [25] focus on proximity detection between friends. Several papers propose Geo-Social recommendation systems [21][27][24]. All these methods are offline data mining tasks that are based on historical data, e.g. users' and their friends' past check-ins.

3 Problem Formulation

A social network can be modeled as an undirected graph $G=(V,E)$, where a node $v_i \in V$ represents a user and an edge $(v_i, v_j) \in E$ indicates the friendship between v_i and $v_j \in V$. A GeoSN is a social graph, where each node may contain the coordinates of the corresponding user.

Let V_i be the *relevant* friends of user v_i for a given query location q. A geo-social ranking function $f(q, v_i)$ assigns to each user v_i a *score* that considers i) the distance $||q, v_i||$ between the query and v_i, ii) the distance between q and the users in V_i, iii) the cardinality of V_i, and iv) possibly the social connectivity of V_i. Factors i) and ii) constitute the *spatial* aspect, whereas iii) and iv) correspond to the *social* aspect.

Definition 1 GSR Top-k query. Given a query point q, a positive integer k, and a GSR function f, return a list R of k tuples $R = (\{v_1, f(q, v_1), V_1\}, \ \ldots, \ \{v_k, f(q, v_k), V_k\})$ such that for each $1 \leq i \leq k$:

$$f(q, v_i) \geq f(q, v_{i+1}) \wedge$$
$$(\nexists \{v_l, f(q, v_l), V_l\} \notin R : f(q, v_k) \leq f(q, v_l))$$

Specifically, the output contains the k users with the highest scores, and their relevant sets, i.e., their friends that participated in the computation of those scores. Ideally, the top-k users should be near q, and have many friends close to the query, possibly tightly connected to each other. As discussed in the running example of Figure 1, different GSR functions are essential because various applications may involve diverse concepts of spatial and social aspects, and employ different ways to combine them for the computation of the total user score.

In the rest of the paper, we propose algorithms that exploit the characteristics of GSR functions to enhance performance. Our algorithms use some social and spatial primitives: i) $GetFriends(v_i)$ returns the friends of v_i and their locations, ii) $GetDegree(v_i)$ returns the number of v_i's friends, iii) $RangeUsers(q, r)$ returns the users within distance r from q, iv) $NearestUsers(q, k)$ returns the k nearest users, and v) $NextNearestUser(q)$ returns incrementally the next nearest user to q.

These operations are easily supported by GeoSNs; e.g., in an adjacency list implementation of the social network, primitives i) and ii) simply involve the retrieval of the friend list of v_i. The efficient processing of iii) to v) necessitates a spatial index, which is already present in most GeoSNs for supporting services like Facebook's *Nearby* and Foursquare's *Radar* (both services return the friends that recently checked-in near the current location of a user). Several implementations of these primitive operations in different architectures are compared in [3]. Although in this paper we assume Euclidean space, the distance metric is orthogonal to the ranking function; we can use other distance functions (e.g., network), if they are supported by the system.

4 Linear Combination

The Linear Combination (LC) of partial scores has been widely used as a ranking function [10][28]. According to LC, the score of a user v_i is the weighted sum of the normalized *social* and *spatial* scores of v_i. For the social score, we consider the number of relevant friends of v_i, whereas for the spatial score we consider their distances to the query point.

4.1 Ranking Function

Given a query q, let $|V_i|$ be the set that contains[1] user v_i and his relevant friends for q. In LC, the social score S_i of v_i is based on the cardinality $|V_i|$, normalized in the range $(0, 1]$. Specifically, $S_i = \frac{|V_i|}{F}$, where $F = dg_{max}+1$ and dg_{max} is the maximum node degree in the social graph. The spatial score G_i of v_i is inversely proportional to the sum of the distances of users in V_i to the query point q. To adjust the spatial score in the range $(0, 1]$, we can divide it with the maximum possible sum of distances defined as $F \cdot C$, i.e., $G_i = 1 - \frac{\sum_{v \in V_i} ||q,v||}{F \cdot C}$, where C is a constant (e.g., it can be the maximum distance between the query and any user). As we will discuss shortly, there are several choices for setting the values of normalization parameters F and C. Independently of these values, Equation 1 describes the LC function, where $w \in (0, 1)$ specifies the relative importance of the social and spatial costs; when $w > 0.5$, the number of relevant friends is more important than their distance to the query.

$$f_{LC}(q, v_i) = w \cdot S_i + (1 - w) \cdot G_i \Rightarrow$$

$$f_{LC}(q, v_i) = w \cdot \frac{|V_i|}{F} + (1 - w) \cdot (1 - \frac{\sum_{v \in V_i} ||q, v||}{F \cdot C}) \quad (1)$$

Given Equation 1, our goal is to compute the relevant set V_i that maximizes $f_{LC}(q, v_i)$. Assume that we consider the inclusion of a friend u_j of v_i in V_i: adding u_j increases the social score by $\Delta S_i = \frac{w}{F}$ and decreases the spatial score by $\Delta G_i = (1-w) \cdot \frac{||q,u_j||}{F \cdot C}$. In order for u_j to be included in V_i, the positive contribution should exceed the negative:

$$\Delta S_i > \Delta G_i \Rightarrow \frac{w}{F} > (1-w)\frac{||q, u_j||}{F \cdot C} \Rightarrow ||q, u_j|| < \frac{w \cdot C}{1 - w}$$

Consequently, the relevant set of of v_i is defined in Equation 2:

$$V_i = \{v_i\} \cup \{u_j : \ u_j \text{ friend of } v_i \wedge ||q, u_j|| < \frac{w \cdot C}{1 - w}\} \quad (2)$$

We refer to the circle centered at q with radius $\frac{w \cdot C}{1-w}$ as the *relevant range*. According to Equation 2, only the users in the relevant range can participate in the relevant sets of their friends. Note that Equation 2 does not restrict v_i, who can be arbitrarily far from q. To overcome this issue, we constrain v_i to be located within the bounds of the relevant range as well. Assuming $w = 0.2$, $C = 30$ (i.e., $\frac{w \cdot C}{1-w} = 7.5$) in the example of Figure 1, users v_1 to v_6 are closer to q than 7.5, and in the relevant range; the rest are ignored. Observe that as opposed to w and C, which determine the relevant users, the parameter F only affects their relative scores. Accordingly, we can re-set F using the maximum number of friends in the relevant range. In this example, the user with the most friends is v_4 with $V_4 = \{v_4, v_1, v_3, v_5\}$, and $F = 4$.

A main weakness of LC is that the result is sensitive to the value of the normalization parameter C. At one extreme, large values of C may lead to a wide relevant range that contains numerous users, several of which are very far from the query. This has negative effect on both the cost of query processing and on the quality of the top-k result since the scores take into account distant friends that are unrelated to the query. At the other extreme, a small relevant range may contain no users; in this case, for each v_i we have $V_i = \{v_i\}$ and $S_i = \frac{1}{F}$ because the contribution of each user to the total score of his friends is negative. Since the social score of all users is constant, their ranking depends only on their distance to q, and the top-k query would degenerate to a conventional k nearest neighbor search.

Ideally, the relevant range should contain a number of users K ($K > k$) large enough to include friends of the top-k users, but at the same time it should preserve

[1] The inclusion of v_i in the relevant set V_i simplifies the problem formulation in LC and RC.

the query locality. To resolve this issue, we propose two approaches for setting C: i) *user-defined* and ii) *data-dependent*. In i), the user explicitly specifies the value of C (e.g., a merchant may be interested in potential customers within $1km$ from his location). In ii), the value of C is set so that the expected number K of users within the relevant range is a function of k (e.g., $K = k^4$). For this estimation, we use a multi-dimensional histogram capturing the user locations and the cost model of [19]. Note that for both approaches, the relevant range is $\frac{w \cdot C}{1-w}$. If $w = 0.5$, this value is the same as C. If $w > 0.5$, the relevant range is expanded with respect to the (user-defined or data-dependent) value of C, to allow the inclusion of friends that are farther than C. On the other hand, if $w < 0.5$ the relevant range shrinks to focus on the users nearest to q.

4.2 Query Processing

Equation 2 leads to a simple and efficient algorithm: perform a range query to find all users within distance $\frac{w \cdot C}{1-w}$ from q. For each of these users, compute the relevant set and score, and return the k users with the highest scores. Figure 2 shows the pseudocode for LC query processing. Line 2 applies the $RangeUsers$ primitive to find all users in the relevant range. Then, for each retrieved user v_i, Lines 4-5 calculate the relevant set (by intersecting the results of primitives $RangeUsers$ and $GetFriends$) and the score. If the total score of v_i exceeds the k-th highest score b_s retrieved so far, v_i is inserted into the result set R, and b_s is updated accordingly. If there are fewer than k users in the relevant range, Lines 9-11 complete the result by incrementally retrieving the nearest neighbors of q.

Input: Location q, positive integer k, weight w, radius C
Output: Result set R // R is sorted on f_{LC} in desc. order

1. $b_s = 0$, $R = \emptyset$
2. $U = RangeUsers(q, \frac{w \cdot C}{1-w})$
3. **For** each $v_i \in U$
4. $\quad V_i = \{v_i\} \cup (GetFriends(v_i) \cap U)$
5. $\quad f_{LC}(q, v_i) = w \cdot \frac{|V_i|}{F} + (1-w) \cdot (1 - \frac{\sum_{v \in V_i} ||q,v||}{F \cdot C})$
6. $\quad$ **If** $f_{LC}(q, v_i) > b_s$
7. $\quad\quad$ add $\{v_i, f_{LC}(q, v_i), V_i\}$ to R
8. $\quad\quad$ b_s = score of the k^{th} tuple in R
9. **While** $|R| < k$
10. $\quad v_i = NextNearestUser(q)$ outside relevant range
11. $\quad$ add $\{v_i, f_{LC}(q, v_i), V_i\}$ to R
12. **Return** R

Fig. 2 LC Top-k Algorithm

We can extend LC to consider the social connectivity of the users in the relevant sets. In particular, given the spatial range C and preferences factor w, the Linear Combination Connectivity (LCC) function considers the same users as LC, i.e., those within distance $\frac{w \cdot C}{1-w}$ from q, and computes identical relevant sets, but it assigns a score to each user v_i based on Equation 3.

$$f_{LCC}(q, v_i) = w \cdot \frac{2 \cdot |E_i|}{F \cdot (F-1)} + (1-w) \cdot (1 - \frac{\sum_{v \in V_i} ||q, v||}{F \cdot C}) \quad (3)$$

where $E_i = \{(u, v) \in E \ : \ u, v \in V_i\}$ and $\frac{2 \cdot |E_i|}{F \cdot (F-1)}$ is the *normalized density* of the graph (V_i, E_i). The only modification to the algorithm in Figure 2 is in line 5, where Equation 3 replaces Equation 1.

5 Ratio Combination

Similar to LC, the Ratio Combination (RC) of two different attributes has been often used in the literature for ranking purposes [22][4]. In GeoSN, the RC score of a user v_i is proportional to the cardinality of the relevant set V_i, and inversely proportional to the sum of distances between q and the users in V_i.

5.1 Ranking Function

We start with a straightforward implementation of RC, explain its shortcomings, and then propose a more general function. Let V_i be the set that contains v_i and his relevant friends; as we will show, V_i is different from the one obtained by LC. According to Equation 4, the RC score of a user v_i is simply the ratio of $|V_i|$ over $\sum_{v \in V_i} ||q, v||$. The usage of multiplicative weights (e.g., $w \cdot |V_i|$) would be meaningless as it simply multiplies each score by a constant, without affecting the rank of the results.

$$f_{RC}(q, v_i) = \frac{|V_i|}{\sum_{v \in V_i} ||q, v||} \quad (4)$$

Consider the inclusion of a friend u_j of v_i in V_i. This addition would increase both the cardinality of V_i (by 1) and the sum of distances (by $||q, u_j||$) yielding a new score for v_i:

$$\frac{|V_i|+1}{||q,u_j|| + \sum_{v \in V_i} ||q,v||}$$

Friend u_j is beneficial for v_i, if and only if, the new score exceeds the old one:

$$\frac{|V_i| + 1}{||q, u_j|| + \sum_{v \in V_i} ||q, v||} > \frac{|V_i|}{\sum_{v \in V_i} ||q, v||} \Rightarrow$$

$$||q, u_j|| < \frac{\sum_{v \in V_i} ||q, v||}{|V_i|}$$

In other words, u_j contributes to the score of v_i, if and only if, his distance to q is less than the average distance of the users currently in V_i. We call the circle centered at q with radius $\frac{\sum_{v \in V_i} ||q,v||}{|V_i|}$ the *current range*[2] of v_i. The current range leads to an intuitive way for computing the relevant set V_i of each user: i) initialize $V_i = \{v_i\}$, ii) retrieve the friends of v_i and sort them in increasing order of their distance from q; iii) incrementally add the sorted users in V_i, until reaching the first friend u_j such that $||q, u_j|| \geq \frac{\sum_{v \in V_i} ||q,v||}{|V_i|}$. User u_j and all subsequent friends in the ordered list are out of the current range; thus, they cannot have a positive contribution to the total score of v_i, and are excluded from V_i.

As opposed to LC, Equation 4 does not involve normalization parameters (e.g., F, C). However, for small values of k, top-k retrieval may still degenerate to k nearest neighbor search. For instance, consider the 4 nearest users to the query q depicted in Figure 3, with distances 1, 2, 3, 4, respectively. The only social connections are between v_1 and v_4, and between v_2 and v_4. Using Equation 4, the relevant sets of the four users are $V_1 = \{v_1\}$, $V_2 = \{v_2\}$, $V_3 = \{v_3\}$, $V_4 = \{v_1, v_2, v_4\}$ and their scores are 1, $\frac{1}{2}$, $\frac{1}{3}$, $\frac{3}{7}$, respectively; i.e., the results of top-k and k-NN search start differentiating for $k > 2$.

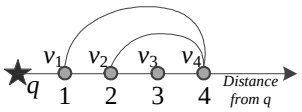


Fig. 3 RC Example

Intuitively, the problem exists because the relevant set V_i cannot contain users with distance larger than $||q, v_i||$. Therefore, the top-1 result is always the nearest user of the query. The subsequent ($k > 1$) top-k users are also likely to be k-NNs of the query, unless there are users farther (e.g., v_4 in the above example) that are friends with those very close to the query. To resolve this issue, we modify the scoring function of RC using Equation 5, where $w \in [0, 1)$ increases with the importance of the social compared to the spatial aspect. As we will show, the subtraction of w from the numerator of the fraction allows the extension of the current range of v_i beyond $||q, v_i||$, differentiating the top-k and k-NN sets based on the social connectivity. If $w = 0$, Equation 5 reduces to Equation 4.

$$f_{RC}(q, v_i) = \frac{|V_i| - w}{\sum_{v \in V_i} ||q, v||} \tag{5}$$

In order to define the new current range, we follow the same approach as in Equation 4; i.e., a friend u_j is beneficial for v_i, if and only if:

$$||q, u_j|| < \frac{\sum_{v \in V_i} ||q, v||}{|V_i| - w} \Rightarrow$$

$$||q, u_j|| < \frac{|V_i|}{|V_i| - w} \cdot \frac{\sum_{v \in V_i} ||q, v||}{|V_i|} \tag{6}$$

The process for computing the relevant sets remains the same, but we use the new current range to define the stopping condition. Specifically, Figure 4 shows the pseudocode of *RC_RelevantSet*, which returns the relevant set of a user v_i. The algorithm takes as input v_i, the query location q, the weight w, and an array A, which contains the distances between the friends of v_i and q, sorted in ascending order. Initially, V_i is set to $\{v_i\}$. Then, friends are added to V_i according to their order in A, until finding the first user u_j, whose distance to q reaches or exceeds $\frac{|V_i|}{|V_i|-w} \cdot \frac{\sum_{v \in V_i} ||q,v||}{|V_i|}$.

Input: user v_i, location q, weight w, sorted array of distances A
Output: Relevant Set V_i

1. $V_i = \{v_i\}$
2. u_j = user with minimum distance in A
3. **While** $||q, u_j|| < \frac{|V_i|}{|V_i|-w} \cdot \frac{\sum_{v \in V_i} ||q,v||}{|V_i|}$
4. $\quad V_i = V_i \cup \{u_j\}$
5. $\quad u_j$ = user with the next distance in A
6. **Return** V_i

Fig. 4 Pseudocode of *RC_RelevantSet*

Compared to Equation 4, the incorporation of w in Equation 5 extends the current range by a factor $\frac{|V_i|}{|V_i|-w}$. Figure 5 plots the value of $\frac{|V_i|}{|V_i|-w}$ as a function of $|V_i|$ and w. The extension is maximized for small $|V_i|$; e.g., if $|V_i| = 1$ and $w = 0.5$, then $\frac{|V_i|}{|V_i|-w} = 2$, so that if v_i has a friend u_j within distance less than $2 \cdot ||q, v_i||$ from q, u_j is added to V_i because this increases $f_{RC}(q, v_i)$. On the other hand, the effect of w diminishes with $|V_i|$; e.g., if $|V_i| = 5$ and $w = 0.5$, then the range is extended by only $\frac{10}{9}$. The intuition is that as the cardinality of V_i increases, we restrict the distance around q where we look for relevant friends of v_i, in order to preserve query locality. In the example of Figure 3, if $w = 0.7$, then the relevant sets of the four users are $V_1 = \{v_1\}$, $V_2 = \{v_2, v_4\}$, $V_3 = \{v_3\}$, $V_4 = \{v_1, v_2, v_4\}$ and their scores are 0.3, $\frac{1.3}{6} = 0.22$, $\frac{0.3}{3} = 0.1$, $\frac{2.3}{7} = 0.33$, respectively.

[2] Note that the current range in RC depends on the friends already in V_i, whereas the relevant range in LC is defined based only on w and C, and it is the same for all users.

Note that the top-1 user is v_4, despite being the 4-th nearest neighbor of q, because of his friendship with v_1 and v_2.

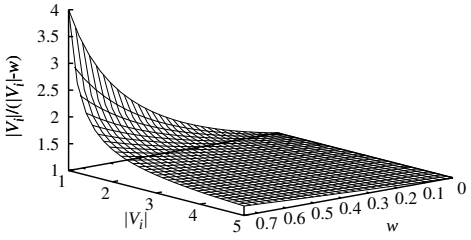


Fig. 5 Range Extension vs. $|V_i|$ and w

Input: Location q, positive integer k, weight w
Output: Result set R sorted on f_{RC} in desc. order

1. $b_s = -\infty$, $T = +\infty$, $R = \emptyset$, $index = 0$
2. **While** $b_s < T$
3. $v_i = NextNearestUser(q)$
4. $N_i = GetFriends(v_i)$
5. FA = sorted array with distances between q and users in N_i
6. $V_i = RC\_RelevantSet(q, v_i, w, FA)$
7. $f_{RC}(q, v_i) = \frac{|V_i| - w}{\sum_{v \in V_i} ||q,v||}$
8. **If** $f_{RC}(q, v_i) > b_s$
9. add $\{v_i, f_{RC}(q, v_i), V_i\}$ to R
10. b_s = score of the k^{th} tuple in R
11. $index = index + 1$
12. **If** $index \leq dg_{max}$
13. $RA[index] = ||q, v_i||$
14. **For** $m = index + 1$ to dg_{max}
15. $RA[m] = ||q, v_i||$
16. $V_{nr} = RC\_RelevantSet(q, v_i, w, RA)$
17. $T = \frac{|V_{nr}| - w}{\sum_{v \in V_{nr}} ||q,v||}$
18. **Return** R

Fig. 6 RC Top-k Algorithm

5.2 Query Processing

Top-k query processing using RC is based on the branch and bound (BnB) framework. Specifically, BnB retrieves users in an iterative manner, computes their score with respect to a function f, maintains the k users with the highest scores, and terminates when the upper bound score T of any unseen user cannot exceed the score b_s achieved by the retrieved users.

Figure 6 illustrates RC query processing. Users are considered incrementally according to their distance from q. For every retrieved user v_i, Lines 3-5 obtain his friends, sort them in ascending order of their distance to q, and insert them in an array FA. Line 6 invokes $RelevantSet$ to compute the relevant set V_i. If the score of v_i exceeds the k-th highest score of the users encountered so far, stored in variable b_s, then v_i is inserted in the result R, and b_s is updated accordingly. The main intricacy refers to the termination condition. Specifically, the incorporation of w in RC complicates the computation of the upper bound score T that a not-yet retrieved user v_{nr} can reach because, if $w > 0$, the relevant range of v_{nr} may extend beyond $||q, v_{nr}||$.

Lines 11-17 in Figure 6 deal with the computation of T. Let dg_{max} be the maximum node degree in the social graph. The distances of the first dg_{max} users (i.e., the dg_{max} users nearest to q) are stored in an array RA, where $RA[index]$ has the distance of the last retrieved user v_i. The upper bound score for any not-yet retrieved user v_{nr} occurs when (i) $||q, v_{nr}|| = ||q, v_i||$ because v_{nr} is at least as far as v_i, and (ii) v_{nr} is a friend with the dg_{max} closest users to q. Based on this, we can compute the best possible relevant set V_{nr} of v_{nr} by invoking $RC\_RelevantSet(q, v_i, w, RA)$. The bound T corresponds to the score achieved by V_{nr}. If the number of retrieved users $index$ is below dg_{max}, Lines 14-15 fill the remaining distances ($RA[index+1]$ to $RA[dg_{max}]$), assuming that all non-retrieved users up to dg_{max} have the same distance to q as the last user v_i.

RC can be extended to capture the connectivity of the relevant sets. Equation 7 presents the Ratio Combination Connectivity (RCC) function, where instead of the cardinality of V_i used in RC, it considers the connectivity of the users in V_i.

$$f_{RCC} = \frac{|E_i| - w}{\sum_{v \in V_i} ||q, v||} \tag{7}$$

The relevant set V_i of v_i deviates from that of RC. Specifically, to compute V_i that maximizes f_{RCC}, we utilize the *maximum weighted densest subgraph* (MWDS) problem. Given a graph with positive vertex weights, MWDS finds the subgraph of maximum weighted density, defined as the number of edges divided by the total weight of the vertices [5]. In our setting, the input graph to MWDS is the induced graph of v_i's friends, where a vertex weight represents the distance of the corresponding user to q. Thus, the relevant set V_i will contain v_i and his friends in the result of MWDS. The RCC processing algorithm is similar to the algorithm of Figure 6, but the $RC\_RelevantSet$ function is replaced by a method for solving MWDS [5]. Finally, lines 7 and 17 compute Equation 7.

6 h-Geo-Social Index

The h-Geo-Social (HGS) ranking function is inspired by the bibliographic h-index. The h-index of an author corresponds to the maximum number h of his papers that have at least h citations [11].

6.1 Ranking Function

Let D_1, D_2, .., D_l,.., be an increasing sequence of positive numbers and group $G_i = \{v_i \cup N(v_i)\}$, where $N(v_i)$ denotes the set of friends of user v_i. The HGS index h_i of v_i is the largest integer l such that $\forall m \in [1, l]$, $\exists$ m members of G_i within distance D_m from q. If such an l does not exist, then $h_i = 0$. The score of a user v_i is equal to his HGS index, i.e., $f_{HGS}(q, v_i) = h_i$, and the relevant set V_i contains v_i and his friends within distance D_l from q.

To set the values of D_1, D_2,.., we use the arithmetico-geometric sequence[3] presented in Equation 8. Parameter w ($w > 0$) adjusts the relative importance of the social and spatial aspects. A large value of w favors social connections since it leads to large ranges and increases the probability that a user has friends within the ranges near the query; w can be set using the *user-defined* and *data-dependent* approaches described in Section 4.

$$D_l = \sum_{b=1}^{l} \frac{w + (b-1) \cdot w}{2^{b-1}} \tag{8}$$

Figure 7 applies HGS ranking to the running example assuming $w = 2.5$. The dashed circles correspond to the ranges defined by D_l for $1 \leq l \leq 5$, i.e., $D_1 = w = 2.5, D_2 = 5, D_3 = 6.9, D_4 = 8.1, D_5 = 8.9$. Note that the difference between consecutive ranges gradually decreases in order to achieve query locality. Consider user v_4, with friends $\{v_1, v_3, v_5, v_7, v_8\}$. Observe that (i) v_4 has exactly one friend in between each of the first 5 rings, i.e., 1 friend within distance D_1 from q, 2 friends within D_2 and so on, up to 5 friends within D_5, and (ii) $||q, u_4|| < D_5$. Thus, $f_{HGS}(q, v_4) = h_4 = 6$. Similarly for user v_2, we have $f_{HGS}(q, v_2) = 2$ since $||q, v_2|| < D_1$ and v_2 has a friend v_1 within distance D_1 from q, but no other friend within D_3.

The ranking function of HGS allows the progressive expansion of V_i, and the corresponding score increase, depending on the number of friends of v_i near the query. Consider for instance that v_i has 2 friends u_1, u_2 within distance D_1. If $||q, v_i|| \leq D_2$, then h_i is at least 3. Assuming that there are 2 more friends u_3, u_4 of v_i farther than D_1 and closer than D_3, then these are also included in V_i and h_i becomes 5. The expansion continues accordingly if there are more friends within distance D_4. Consequently, HGS is preferable for applications that benefit from this progressive behavior,

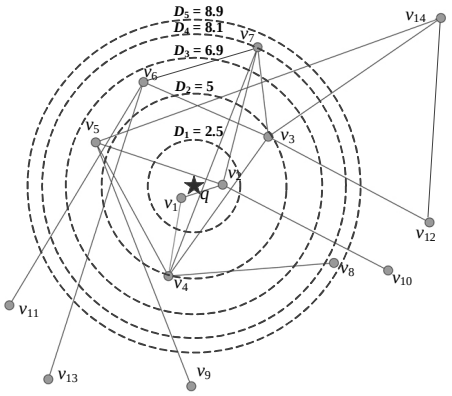


Fig. 7 HGS Ranges

i.e., favor the inclusion of friends far from the query for users who have many friends in the proximity.

6.2 Query Processing

According to the definition of HGS, the only users with a non-zero score, are those who either are within distance D_1 from q, or have at least a friend in this range. Based on this observation, top-k query processing using HGS involves two phases. Phase 1 performs a range query to find users within distance D_1, and retrieves their friends within distance $4 \cdot w$. The users outside this range cannot participate to the result because of the arithmetico-geometric sequence[4]. These users constitute the candidate results. Phase 2 computes the HGS indexes of the candidates and returns the k users with the largest indexes. Figure 8 illustrates the pseudocode of HGS top-k retrieval.

Lines 2-7 correspond to phase 1, i.e., generation of the candidate set. Lines 8-21 perform HGS computation: for each candidate v_i, the algorithm sorts v_i and his friends in ascending order of their distance to q, and inserts the sorted distances in an array FA. Starting with $l = 1$ (i.e., $D_1 = w$), while the l-th distance in FA is less than D_l, the corresponding user is inserted in V_i because v_i has at least l friends within D_l from q. When the loop of Lines 12-14 terminates ($||q, FA[l]|| > D_l$), the score of v_i is set to $|V_i|$ provided that v_i is also in the distance range. If the score of v_i exceeds the current best (b_s), the result set R and b_s are updated accordingly. Finally, if the result contains fewer than k users, Lines 18-20 complete it by incrementally retrieving the nearest neighbors of q, who are not already in R.

HGS ranks users based on the number of friends located in concentric rings around the query point. Its nature renders the incorporation of connectivity infor-

[3] An arithmetico-geometric sequence is the result of the multiplication of a geometric progression with the corresponding terms of an arithmetic progression. The sequence exhibits geometric decay and approaches a maximum value of $4 \cdot w$, i.e. $\lim_{b \to \infty} D_b = 4 \cdot w$. Other series (e.g., arithmetic, geometric) can also be applied.

[4] This optimization is specific to Equation 8. Other bounds would apply for different series.

Input: Location q, positive integer k, weight w
Output: Result set R

1. $R = \emptyset$, $b_s = 0$
2. $U = RangeUsers(q, w)$
3. $Candidates = U$
4. **For** each $v_i \in U$
5. **For** each $u \in GetFriends(v_i)$
6. **If** $||q, u|| \leq 4 \cdot w$
7. $Candidates = Candidates \cup \{u\}$
8. **For** each $v_i \in Candidates$
9. $V_i = \emptyset$
10. FA = array of v_i and his friends sorted on their distances to q
11. $l = 1$, $D_l = w$
12. **While** $||q, FA[l]|| \leq D_l$
13. $V_i = V_i \cup FA[l]$
14. $l = l + 1$, $D_l = D_l + \frac{w+(l-1)\cdot w}{2^l - 1}$
15. $f_{HGS}(q, v_i) = |V_i|$
16. **If** $f_{HGS}(q, v_i) > b_s$
17. update R and b_s
18. **While** $|R| < k$
19. $v_i = NextNearestUser(q)$
20. **If** $v_i \notin R$ **Then** add v_i to R
21. **Return** R

Fig. 8 HGS Top-k Algorithm

mation among friends meaningless. However, the density of V_i could be used as an additional criterion for ordering users with the same HGS score.

7 Geo-Social Triangles

Geo-Social Triangles (GST) is motivated by the Geographic Clustering Coefficient [16], which combines the social clustering coefficient with spatial criteria, and the Average Triangle Length metric [17], which is the average length of the triangles that a user forms with his friends. Specifically, the GST ranking function takes into account the friends that a user and his friends have in common so that his score is based on the number of triangles in which he participates, and their distances from the query point.

7.1 Ranking Function

Let u_j, u_p be two friends of v_i. If u_j, u_p are also friends with each other, then v_i, u_j, u_p form a triangle. The score of a triangle is based on the distances of its members to q, i.e., $||q, v_i||$, $||q, u_j||$, and $||q, u_p||$. The score of v_i is the sum of the individual scores of the triangles in which he participates. The GST function in Equation 9 assigns comparable scores to triangles close to q, and exponentially lower scores to triangles with large total distances. Therefore, the top-k users are those with many triangles near q. The relevant set V_i contains all users that form triangles with v_i.

$$f_{GST}(q, v_i) = \sum_{\text{triangle } v_i, u_j, u_p} e^{-\frac{||q,v_i||+||q,u_j||+||q,u_p||}{w}} \quad (9)$$

Parameter w ($w > 0$) adjusts the relative importance of the social and spatial aspects. In particular, as w increases, the value of fraction $\frac{||q,v_i||+||q,u_j||+||q,u_p||}{w}$ decreases and, due to the exponential function, the scores of triangles with different total distances start converging to the same value. Consequently, the importance of the proximity to q decreases, favoring users with numerous triangles, even if they are far from the query.

For instance, consider two users, v_1 and v_2, where v_1 participates in exactly one triangle with total distance 2, and v_2 is a member of two triangles each having total distance 2.1. If $w = 0.1$, then the scores of v_1 and v_2 are $2.2 \cdot 10^{-9}$ and $1.6 \cdot 10^{-9}$, respectively. On the other hand, when $w = 1$, the score of v_1 (0.13) is lower than that of v_2 (0.24).

7.2 Query Processing

Query processing is based on a branch and bound algorithm that generates a candidate set by only considering triangles near the query. The candidates are then refined to produce the top-k users. Figure 9 describes GST top-k query processing. Users are retrieved in ascending order of their distances to query point q. Let v_i be the last user; $||q, v_i||$ is inserted in an array RD that contains the distances of the retrieved users. $SC[v_i]$ and $TR[v_i]$ maintain the current score and number of triangles involving v_i, respectively. Lines 5-6 obtain the friends of v_i, sort them in ascending order of distance to q, and insert those closer to q than v_i in a sorted list NL (i.e., NL only includes users that have already been retrieved). Lines 7-16 form all triangles that contain v_i and his friends in NL; i.e., triangles are discovered in a *lazy* way, when the farthest of the three nodes is encountered.

Specifically, for each pair u_j, u_p of users in NL, if u_j and u_p are friends, the score of the new triangle is computed, and the scores and counters of v_i, u_j, u_p change accordingly. The current top-k result CR, best score b_s and upper bound T are also updated. The iterative examination of users terminates when the best possible score of any user is below the current best. Note that the score of v_i (and all retrieved users) is potentially incomplete because it does not consider triangles containing v_i and some user farther than $||q, v_i||$. In general, this approach avoids triangles far from q that have exponentially small scores. However, it necessitates a refinement step (Line 17) to complete the scores for the candidate results by finding their remaining triangles.

Before proceeding to the refinement step, we discuss the computation of the upper bound T. Let tr_{max} be the maximum number of triangles in which any user participates (tr_{max} is query-independent and can be

Input: Location q, positive integer k, weight w
Output: Result set R

1. $b_s = 0$, $T = +\infty$, $CR = \emptyset$,$RD = \emptyset$, $SC = \emptyset$, $TR = \emptyset$, $index = 0$
2. **While** $b_s < T$
3. $v_i = NextNearestUser(q)$
4. $RD[++index] = ||q, v_i||$, $SC[v_i] = 0$, $TR[v_i] = 0$
5. $N_i = GetFriends(v_i)$
6. NL = sorted users of N_i with distance $\leq ||q, v_i||$
7. **For** $j = 1$ to $|NL| - 1$
8. $u_j = NL[j]$
9. **For** $p = j + 1$ to $|NL|$
10. $u_p = NL[p]$
11. **If** u_j and u_p are friends
12. $s = e^{-\frac{||q,v_i||+||q,u_p||+||q,u_j||}{w}}$
13. $SC[v_i] = SC[v_i] + s$, $TR[v_i] = TR[v_i] + 1$
14. $SC[u_j] = SC[u_j] + s$, $TR[u_j] = TR[u_j] + 1$
15. $SC[u_p] = SC[u_p] + s$, $TR[u_p] = TR[u_p] + 1$
16. update CR, b_s and T
17. $R = GST\_refinement(q, w, k, ||q, v_i||, CR, SC, TR, RD)$
18. **Return** R

Fig. 9 GST Top-k Algorithm

computed off-line in $O(|V|^{2.6})$ [12]). The best score that a non-retrieved user v_{nr} can obtain is the sum of the scores of the tr_{max} highest scoring triangles that can contain v_{nr}. Intuitively, these triangles consist of v_{nr} and the closest users to q. Based on this observation, we construct, at the beginning of the algorithm, the array BT of size tr_{max} that contains the distances of the tr_{max} pairs of users with the minimum sum of distances to q. For example, let us assume that $tr_{max} = 4$ and the four closest users to q have distances 1, 2, 3, and 4.5, respectively. In this case, we have $BT = \{1 + 2, 1 + 3, 2 + 3, 1 + 4.5\}$. Then, we can simply compute T by summing up the scores of the tr_{max} triangles using $||q, v_{nr}||$ and the distances in BT. For instance, the score of the l-th best triangle is $e^{-\frac{||q,v_{nr}||+BT[l]}{w}}$.

Figure 10 shows the pseudocode of $GST\_refinement$ that implements the refinement step. In addition to q, w, k, the input consists of the current result set CR, the score SC, counter TR and distance RD arrays, and the distance d_{last} of the last retrieved user. Initially, for each user v_i in the current result set CR, the algorithm completes the score by retrieving all triangles involving v_i (i.e., those also containing non-yet retrieved users). The process is similar to Lines 7-16 of Figure 9, but this time without a distance bound. Then, it adds v_i in the (final) result R and updates the k-th best score b_s.

Lines 5-14 deal with the score computation for users, not already in R, who participate in at least one retrieved triangle. The rest of the users can be safely eliminated since their best possible score is below the bound T. For such a candidate user v_i, it should hold that $TR[v_i] > 0$. Before retrieving all triangles of v_i, Lines 6-11 determine an upper bound B_i for his score. Let dg_i be the degree of v_i; the maximum number of triangles containing v_i is $M_i = \min\{tr_{max}, \frac{dg_i \cdot (dg_i - 1)}{2}\}$. B_i is initialized to the current score $SC[v_i]$. The number

Input: q, w, k, distance d_{last}, arrays: CR, SC, TR, RD
Output: Result set R

1. $R = \emptyset$
2. **For** each $v_i \in CR$
3. Complete the score of v_i by retrieving all triangles of v_i
4. Update R and b_s
5. **For** each v_i such that $TR[v_i] > 0 \wedge v_i \notin R$
6. $dg_i = GetDegree(v_i)$
7. $M_i = \min\{tr_{max}, \frac{dg_i \cdot (dg_i - 1)}{2}\}$
8. $B_i = SC[v_i]$, $j = 1$
9. **While** $TR[v_i] \leq M_i$
10. $B_i = B_i + e^{-\frac{||q,v_i||+||q,RD[j]||+d_{last}}{w}}$
11. $j = j + 1$, $TR[v_i] = TR[v_i] + 1$
12. **If** $B_i > b_s$
13. Complete the score of v_i by retrieving all triangles of v_i
14. Update R and b_s
15. **Return** R

Fig. 10 Pseudocode of $GST\_refinement$

of non-retrieved triangles of v_i can reach $M_i - TR[v_i]$; in the best case, each such triangle includes a retrieved user near the query and a non-retrieved user at distance exactly d_{last}. For instance, the distance of the first such hypothetical triangle depends on: i) $||q, v_i||$ because of v_i, ii) $||q, RD[1]||$ because of the nearest user to q (retrieved), and iii) d_{last} because of the non-retrieved user. Based on this, Line 10 computes the score of the l-th best triangle, which contains the l-th nearest user to q. If B_i exceeds the current best score b_s, the actual score of v_i is computed; otherwise, v_i is discarded.

8 Qualitative Evaluation

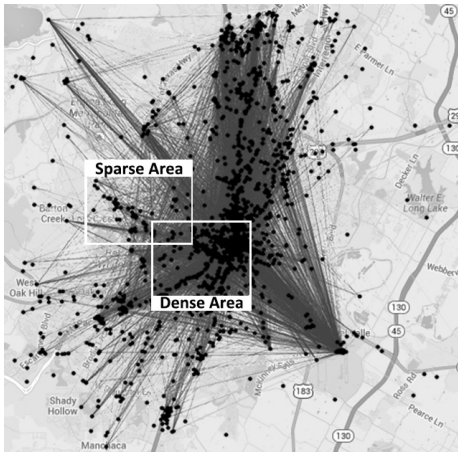


Fig. 11 Check-ins and Social Network

We qualitatively evaluate the behavior of the four ranking functions using a real dataset from Gowalla that includes a social graph and multiple check-ins for each user. We keep only the last check-in of the 5,868 users on March 17th, 2010 in Austin (Texas, US). The average degree in the social graph is 7.6, and the maximum degree is 390. The largest distance between any

two users is $32km$. Figure 11 depicts the check-ins of the users as black dots, and their social connections as grey edges. Section 8.1 visualizes the top-k users, Section 8.2 evaluates the effect of the function parameters on the results, Section 8.3 measures ranking functions' correlation using Kendall's τ_b rank coefficient, and Section 8.4 discusses the suitability of functions to different application scenarios.

8.1 Visualization

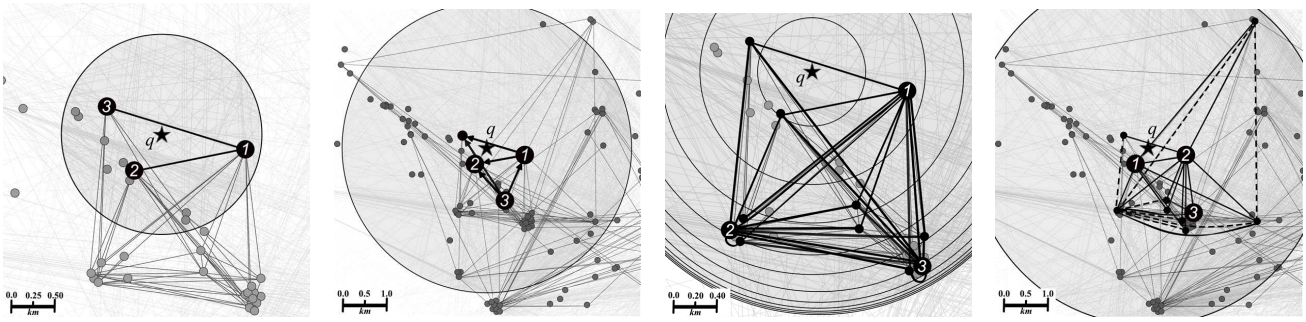
In all the following visualizations, we illustrate the top-3 users of different functions in the sparse and dense areas of Figure 11. For each setting, the query location is the same and represented by a star shape. The top-3 users are depicted as black points together with their rank. A bold edge indicates the social connection between a top user and a *relevant* friend. Relevant friends are drawn as small black points. To facilitate comparison of results by different functions, we use u_{f-i} to denote the top-i user of function f, e.g., u_{LC-1} corresponds to the top-1 result of LC, u_{HGS-2} to the second best user of HGS and so on.

Figure 12(a) depicts the result of the LC function in the sparse area, assuming $w = 0.5$ (i.e., equally important social and spatial scores) and $C = 1km$ (user-defined). The relevant range is the circle centered at q with radius $\frac{w \cdot C}{1-w} = 1km$. The top-1 user u_{LC-1} is farther than u_{LC-2} and u_{LC-3} (distances $0.93km$, $0.46km$, $0.65km$, respectively) because he has 2 friends (u_{LC-2}, u_{LC-3}) in the range, whereas the other users have only 1 (u_{LC-1}). Figure 12(b) repeats the experiment using the RC function and $w = 0.5$. Observe the different scale of the diagrams, illustrated in the bottom left corner. The top-2 users are the same as LC, but u_{RC-3} is different from u_{LC-3}. Specifically, u_{RC-3} is out of the relevant range (distance $1.39km$), and therefore not considered at all by LC. On the other hand, he out-ranks u_{LC-3} in RC due to his friendship[5] with u_{RC-1} and u_{RC-2}. The circle centered at q with radius $3.34km$ contains all users retrieved by RC; the rest are eliminated by the BnB search because they cannot reach the score of u_{RC-3}.

Figure 12(c) presents the results of HGS with $w = 0.5km$. The concentric circles correspond to the ranges of the arithmetico-geometric sequence (the outermost ring is at $2km$). The top-1 user is the same as that of the previous functions because he has the maximum HGS index (10). On the other hand, u_{HGS-2} and u_{HGS-3} appear for the first time; despite both being relatively far from q (distance $1.53km$ and $1.9km$, respectively), they have high HGS indexes (9 and 8), i.e., they could potentially influence numerous users not too far from the query. Figure 12(d) focuses on the GST function assuming $w = 1$. For the sake of clarity, we only show the triangles in the range defined by the distance of the last retrieved user before the refinement step (i.e., all users/nodes of these triangles are in the circular range of Figure 12(d)). The dotted lines connect the two friends of a top user, "closing" a triangle. The top users participate in a similar number of triangles and their scores differ due to the triangle distances. Note that although the actual user distances ($0.46km$, $0.93km$, $1.53km$ for u_{GST-1}, u_{GST-2}, and u_{GST-3}) are not explicitly considered in the scores, users near q are likely to yield smaller triangle distances than farther ones.

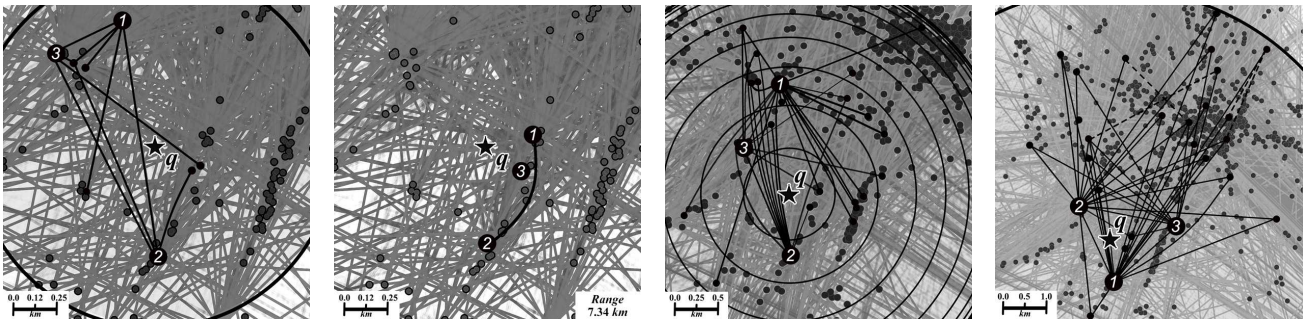
Figure 13 repeats the visualizations for the dense area using the same parameters for all functions. As shown in Figure 13(a), for LC the relevant range contains numerous users; thus, relevant sets are considerably larger compared to the sparse area. Specifically, u_{LC-1} (distance $0.79km$) has 5 friends, u_{LC-2} (distance $0.66km$) has 4 friends, and u_{LC-3} (distance $0.85km$) has 4 friends. On the other hand, for RC (Figure 13(b)) the relevant sets are very small: only u_{RC-1} (distance

[5] A directed bold edge from v_i to v_j means that $v_j \in V_i$.



(a) LC:$w = 0.5, C = 1km$ (b) RC: $w = 0.5$ (c) HGS: $w = 0.5km$ (d) GST: $w = 1$

Fig. 12 Top-3 Users in Sparse Area



(a) LC: $w = 0.5$, $C = 1km$ (b) RC: $w = 0.5$ (c) HGS: $w = 0.5km$ (d) GST: $w = 1$

Fig. 13 Top-3 Users in Dense Area

$0.34km$) and u_{RC-2} (distance $0.52km$) contribute to the score of each other. Recall that if $w = 0.5$, the current range of a user is extended by a factor of 2. Because in dense areas the nearest users (e.g., u_{RC-1}) are very close (e.g., $0.34km$) to the query, it is likely that even the extended range will contain very few of their friends. Therefore, RC prefers locality to social connectivity.

For HGS (Figure 13(c)), the scores of the top users are 16, 10, 8, and their distances are $1.21km$, $0.66km$ and $0.76km$, respectively. The behavior of HGS is rather opposite to that of RC since it favors users that may be relatively distant, but have many friends in the proximity of the query. According to GST, the top users are at distances $0.76km$, $0.79km$ and $0.66km$ and participate in 591, 299 and 365 triangles, respectively. Although u_{GST-3} is closer and has more triangles than u_{GST-2}, the friends participating in these triangles are farther than those of u_{GST-2}. For clarity, Figure 13(d) includes a small subset of the triangles. Finally note that, whereas for sparse areas the results of different functions have large overlap (e.g., $u_{GST-1} = u_{LC-2}$, $u_{GST-2} = u_{LC-1}$ and $u_{GST-3} = u_{HGS-2}$), for dense areas they exhibit high variability because there are numerous users, and thus more choices for the selection of the top results.

8.2 Function Parameters

All functions involve a parameter w used to adjust the relative importance of the social and spatial aspects. Although w is different for each function, in general, increasing its value favors the social connectivity at the expense of query locality. Moreover, LC involves the additional parameter C that defines the relevant range around q. In the sequel, we discuss how the values of w and C affect the top-k results. For consistency with the visualization experiments, we set $k = 3$. All distances are shown in kilometers.

Table 1 LC - Effect of K ($w = 0.5$)

	K	# US	C	# FR (Dist)	# FR (Dist)	# FR (Dist)
Sparse	k^3	27	1.59	8 (0.93)	7 (1.53)	7 (1.392)
	k^4	122	2.99	11 (0.93)	11 (1.53)	11 (1.9)
	k^5	232	4.92	14 (0.93)	14 (0.46)	13 (0.93)
	k^6	878	6.68	59 (3.34)	31 (5.98)	27 (6.4)
Dense	k^3	27	0.44	0 (0.29)	0 (0.29)	0 (0.29)
	k^4	69	0.65	1 (0.34)	1 (0.52)	1 (0.43)
	k^5	266	1.02	5 (0.79)	4 (0.66)	4 (0.85)
	k^6	584	1.94	18 (1.59)	18 (1.19)	16 (1.31)

We start with LC and C. Recall that C can be set explicitly by the user, or it can be computed by a cost model [19] so that the expected number K of users within the relevant range is a function of k. For this experiment, we follow the second approach. Specifically, we set C so that K equals k^3, k^4, k^5 and k^6 ($k = 3$). For each value of K, Table 1 shows the computed value of C, and the actual number (#US) of users within distance C from q. The last three columns contain the number of relevant friends (#FR) and the distance for the top-3 users, respectively. In the first (second) half of the table, the query point is the same as the one used for the sparse (dense) area of the visualization experiments. The value of w is set to 0.5.

Predictably, the relevant range, and consequently the number of relevant users increases with K. In order to comprehend the difference between sparse and dense areas, let us consider that the target number of relevant users is $K = 3^5 = 243$. Naturally, the computed value of C for the sparse area ($4.92km$) is larger than that ($1.02km$) for the dense area. Although in both cases, the actual number of users, 232 and 266 respectively, is similar and close to the expected 243, the relevant sets of the top users have rather different cardinality. Specifically, in the sparse area the top users have 14, 14, and 13 relevant friends, whereas in the dense area the corresponding numbers are 5, 4, and 4. This can be explained by the fact that social connections exhibit higher locality in sparse areas (e.g., neighbors in the

Table 2 LC - Effect of w ($C = 1km$)

	w	# US	RR	# FR (Dist)	# FR (Dist)	# FR (Dist)
Sparse	0.1	0	0.1	0 (0.46)	0 (0.5)	0 (0.56)
	0.3	0	0.42	0 (0.46)	0 (0.5)	0 (0.56)
	0.5	11	1	2 (0.93)	1 (0.46)	1 (0.65)
	0.7	56	2.3	11 (0.93)	11 (1.53)	11 (1.9)
	0.9	3773	9	243 (8.3)	215 (3.34)	204 (8.47)
Dense	0.1	0	0.1	0 (0.29)	0 (0.29)	0 (0.29)
	0.3	22	0.42	0 (0.29)	0 (0.29)	0 (0.29)
	0.5	265	1	5 (0.79)	4 (0.66)	4 (0.85)
	0.7	1244	2.3	54 (1.31)	40 (1.59)	40 (2.23)
	0.9	4108	9	253 (2.66)	234 (5.64)	224 (4.66)

Table 3 RC - Effect of w

	w	# US	BR	# FR (Dist)	# FR (Dist)	# FR (Dist)
Sparse	0.1	8	0.92	0 (0.46)	0 (0.5)	0 (0.56)
	0.3	27	1.7	0 (0.46)	0 (0.5)	2 (0.93)
	0.5	133	3.34	2 (0.93)	2 (0.46)	3 (1.39)
	0.7	166	3.84	2 (0.93)	3 (1.39)	3 (1.4)
	0.9	196	4.43	2 (0.93)	3 (1.39)	3 (1.4)
Dense	0.1	21	0.4	0 (0.29)	0 (0.29)	0 (0.29)
	0.3	525	1.78	0 (0.29)	0 (0.29)	0 (0.29)
	0.5	3875	7.43	1 (0.34)	1 (0.52)	0 (0.29)
	0.7	5515	15	1 (0.34)	1 (0.52)	2 (0.66)
	0.9	5866	23.02	3 (0.66)	1 (0.34)	1 (0.52)

Table 4 HGS - Effect of w

	w (km)	# US	HGS_1 (Dist)	HGS_2 (Dist)	HGS_3 (Dist)
Sparse	0.5	26	11 (0.93)	10 (1.53)	9 (1.9)
	1	57	16 (1.53)	16 (0.46)	15 (0.93)
	1.5	78	32 (3.34)	18 (5.51)	18 (1.53)
	2	179	161 (3.34)	90 (7.9)	77 (7.52)
	2.5	184	233 (3.34)	136 (7.9)	130 (9.94)
Dense	0.5	199	18 (1.21)	12 (0.66)	11 (0.76)
	1	1144	227 (2.66)	124 (3.09)	102 (2.6)
	1.5	1949	242 (2.66)	201 (5.64)	196 (4.66)
	2	2239	250 (2.66)	230 (5.64)	218 (4.66)
	2.5	3807	261 (2.66)	252 (5.64)	232 (4.66)

suburbs), while dense areas (e.g., downtown) are more likely to contain unconnected users.

Another interesting observation is that the distance of the top-3 results increases with the relevant range. For instance, in the sparse area, when $C = 1.59km$, the distances of the top-3 users are $0.93km$, $1.53km$, $1.392km$; when $C = 6.68km$, the corresponding distances are $3.34km$, $5.98km$, $6.4km$. This happens because the number of friends, and therefore the social scores of some users, not necessarily near the query, increases significantly due to the range expansion. In this example, the top users for $C = 6.68km$, have 59, 31 and 27 friends, whereas the top users for $C = 1.59km$ have only 7, 6, and 6 friends.

Table 2 investigates the effect of w ($w \in (0, 1)$) in LC, for $C = 1km$. The #US column contains the number of users within distance C from q and the RR column refers to the relevant range (in km) computed as $\frac{w \cdot C}{1-w}$. Recall that small values of w favor locality. Consequently, for $w \leq 0.3$, LC degenerates to nearest neighbor search in both the sparse and dense areas (note that the top users have zero friends). At the other extreme, $w = 0.9$ increases the relevant range to $9km$, favoring users with numerous friends in the range. For the same relevant range, the top users in the dense area have more friends than those in the sparse area.

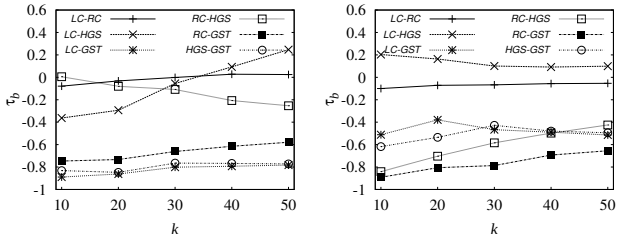
Table 3 studies the effect of w ($w \in [0, 1)$) on RC. The #US column shows the total number of users examined by the BnB technique, and the BR column refers to the corresponding range, i.e., the distance (in km) of the last retrieved user. Similar to LC, for small values of w, top-k retrieval degenerates to k-NN in both the sparse and the dense areas. As opposed to LC, however, increasing the value of w does not have a significant effect on the number of relevant friends, which does not exceed 3 even for $w = 0.9$ because RC favors locality.

Table 4 focuses on HGS, where w ($w > 0$) determines the values of the arithmetico-geometric sequence. The #US column refers to the total number of users examined by the query processing algorithm. HGS_i is the score of the top-i user. Even for the smallest value $w = 0.5km$ in the sparse area, the HGS indexes of the top users are 11, 10 and 9. Recall that $D_1 = w = 0.5km$ and $D_2 = 1km$, whereas the distances of these users are $0.93km$, $1.53km$, and $1.9km$. Although none of the top-3 users is within $0.5km$ from q, they all have at least a friend within $0.5km$, and two friends within $1km$. The HGS indexes increase with w, reaching up to 260 for the top user in the dense area, if $w = 2.5km$. This user is the best result for all values with $w \geq 1$, but with different HGS index in each case. Similar to the values of HGS indexes, the average distances of the top users also increase with w because distant users, who have many friends near q, may become part of the result.

Table 5 investigates the effect of w ($w > 0$) on GST. The #US column contains the total number of users examined by the BnB technique, and the BR column refers to the distance of the last retrieved user, before the refinement step. The last three columns illustrate the number of triangles (#TR) containing the top-3 users and the average distance (AD) of these rectangles. Large values of w reduce the importance of the distance of individual triangles, favoring users with numerous triangles, even if they are relatively far from the query. This explains why both the number of triangles and their average distance increase with w in the dense area, for the top-3 users. On the other hand, for the sparse area, the result is insensitive to w because there are some users in the vicinity of the query that participate in many more triangles than the rest.

Table 5 GST - Effect of w

	w	# US	BR	# TR (AD)	# TR (AD)	# TR (AD)
Sparse	0.5	56	2.59	129 (7.1)	104 (6.45)	54 (6.66)
	1	99	2.89	129 (7.1)	104 (6.45)	124 (7.4)
	1.5	172	4.03	129 (7.1)	104 (6.45)	124 (7.4)
	2	329	5.51	129 (7.1)	104 (6.45)	124 (7.4)
	2.5	980	6.95	129 (7.1)	104 (6.45)	124 (7.4)
Dense	0.5	2713	3.2	299 (8.89)	365 (11)	591 (12.1)
	1	3369	4.93	591 (12.1)	299 (8.89)	365 (11)
	1.5	3631	6.11	1869 (14.2)	1135 (12.21)	591 (12.1)
	2	3849	7.17	1869 (14.2)	1135 (12.21)	591 (12.1)
	2.5	3989	7.89	2624 (15.4)	1869 (14.2)	1795 (14.73)



(a) Sparse Area (b) Dense Area

Fig. 14 GSR functions correlation (Kendall's τ_b vs. k)

8.3 Rank Correlation

Kendall rank correlation coefficients [13] have been widely used to measure the statistical dependence between two ranking functions. Let R_1 and R_2 be the top-k results of two ranking functions on the same query. Kendall's coefficients require that R_1 and R_2 rank the same set of users. Since this may not be true in our setting, we append the missing users at the end of each result set. For example, suppose that $R_1 = \{u_1, u_2, u_3\}$ and $R_2 = \{u_2, u_4, u_5\}$. To relate these results, we set $R_1 = \{u_1, u_2, u_3, u_4, u_5\}$ and $R_2 = \{u_2, u_4, u_5, u_1, u_3\}$. Additionally, we assume that the users appended to a ranking list are assigned the same ordering value, e.g., users u_1 and u_3 are both ranked at the 4th position in R_2.

In our evaluation, we utilize Kendall's τ_b ranking coefficient that supports duplicate ranks, i.e., two users can have the same ranking [6]. τ_b takes into consideration the number of *concordant* pairs (u_i, u_j), i.e., all pairs where u_i and u_j have the same order in both R_1 and R_2, and the number of *discordant pairs*, where their order is different (e.g., u_i is ranked higher than u_j in R_1, but lower in R_2). The coefficient is in the range $-1 \leq \tau_b \leq 1$: i) -1 indicates that the rankings are reverse of each other, i.e., all user pairs are discordant, ii) 0 implies independence of the two ranking functions, i.e., equal number of concordant and discordant pairs and iii) 1 implies that the ranking functions are in perfect agreement, i.e., all pairs are concordant. The correlation among different ranking functions is not transitive, i.e., if ranking function f_1 has a positive (or negative) value of τ_b with f_2, and f_2 with f_3, then this does not imply that f_1 has a positive (or negative) correlation with f_3.

Figures 14(a) and 14(b) plot τ_b for all pairs of ranking functions versus k in sparse and dense areas, respectively. The values of the function parameters are the same as those used in the visualization experiments, i.e., LC: $w = 0.5$, $C = 1km$, RC: $w = 0.5$, HGS: $w = 0.5km$, and GST: $w = 1$. For sparse areas and $k > 30$, LC and HGS have a positive correlation because there are fewer than k users within the query proximity, and both algorithms complete the result set using nearest neighbor search. In dense areas, they have more concordant pairs since users with many friends within the relevant range have high HGS and LC scores. HGS is anti-correlated to RC because it favors the inclusion of friends far from the query point. LC and RC are almost independent in both sparse and dense areas. Moreover, the top-k results of LC and RC have some similarities (i.e., concordant user pairs) since both favor users close to the query point. However, RC prefers locality to social connectivity, which results in differences at the top-k results (i.e., discordant user pairs). The combinations involving GST are the most negatively correlated because GST is the only function that considers inter-connectivity. The negative and zero correlation among functions indicates their unique characteristics and justifies the need for different functions to accommodate diverse application requirements.

8.4 Summary

The presence of parameter C in LC can be both a drawback and an advantage. On the one hand, it may arbitrarily eliminate potentially good results. For instance, in Figure 12(a), for $C = 1km$, the top-1 user has distance $0.93km$. Consequently, if C were below 0.93 this user would not be retrieved. Moreover, a very small value may reduce top-k retrieval to k-NN search, whereas a large value may lead to irrelevant users. On the other hand, C provides a natural way to express real-life constraints, such as the fact that an advertiser is only interested in users within a range; e.g., a restaurant sending lunch promotions to potential customers within $0.5km$.

In RC, friends are included in the relevant set of a user only if they can decrease the average distance to the query point. Consequently, each inclusion impedes the addition of more friends because it tightens the range. Thus, even if a user has many friends in the

proximity of the query, only the few closest ones are considered in his score (in Table 3, the number of relevant friends for the top users is at most 3). Accordingly, RC can be used in cases where locality is crucial. For instance, a cinema has empty seats for a film starting soon, and sends coupons to users in close proximity. Friends of these users are relevant, only if they are also very near.

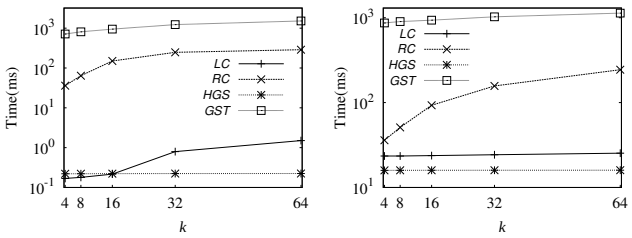
On the other hand, HGS favors the inclusion of friends far from the query for users who have many friends in the vicinity. However, HGS also has a drawback: assume that there are two users with the same HGS index 10. The friends of the first user are evenly spread through the 10 ranges, while those of the second one are concentrated near the query. Although both users have the same score, the second should be preferable[6] because the average distance of his friends is much smaller. HGS is useful for cases where the distance aspect is not critical; e.g., a concert promotion targeting users with many friends in the wider area of the concert.

As opposed to the other functions that consider only the number of relevant friends, GST explicitly takes into account the connectivity of friends in the form of triangles. Locality is measured using the distances of triangles, instead of the individual users. GST is suitable for applications where this connectivity is essential or desirable for ranking; e.g., a promotion similar to that of the concert, but this time for an event (party) that involves social interaction among the various users. Finally, performance criteria may also play a role in the selection of the appropriate function because, as we show in the next section, query processing techniques may involve substantially different costs.

9 Performance Evaluation

The proposed algorithms were implemented in C++ under Linux Ubuntu and executed on an Intel Xeon E5-2660 2.20GHz with 96GB RAM. All data are stored in the main memory. The social graph is kept as a hash table (in the form of key-value pairs), wherein each key is a user id and the value is an adjacency list with the user's friends ids. The locations of users are maintained by a regular spatial grid. The implementation of primitive operations is based on the framework of [3] for centralized, main-memory architecture. Specifically, social primitives perform look-up operations on the hash-table, while spatial primitives execute range and nearest neighbor queries on the spatial grid. Section 9.1 evaluates the efficiency of our methods using the real dataset described in Section 8; Section 9.2 focuses on scalability issues using synthetic data. We report the average values over 20 executions with random query locations.

[6] An analogy for the conventional h-index is two authors that have the same h-index, but the second has more citations.



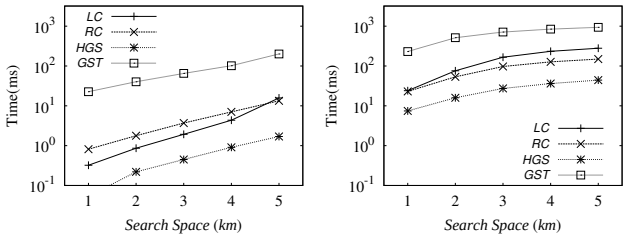
(a) Sparse (b) Dense

Fig. 15 Execution Time vs. k (Real Data)

9.1 Real Data

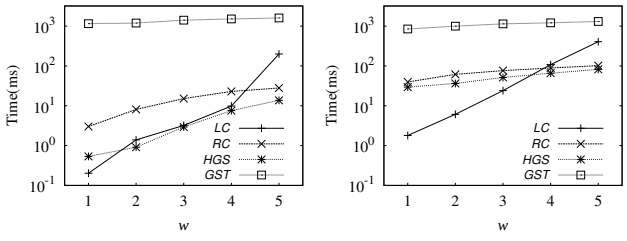
Figure 15 assesses the query time (in milliseconds) as a function of the result set size k in the sparse and dense areas of the real dataset. The values of the function parameters are the same as those used in the qualitative evaluation, i.e., LC: $w = 0.5$, $C = 1km$, RC: $w = 0.5$, HGS: $w = 0.5km$, and GST: $w = 1$. LC and HGS outperform RC and GST in all cases. The value of k does not have a significant impact on their performance because they are based on range queries, except for LC in the sparse area where top-k retrieval reduces to k-NN search for large values of k. On the other hand, the execution time of RC and GST increases with k, since the kth best score decreases, and consequently more users need to be examined by the branch and bound framework. GST is consistently the most expensive because it examines the adjacency list of all the friends for each retrieved user. In dense areas, the performance of HGS and LC deteriorates since their relevant ranges contain numerous users, whereas RC and GST are not seriously affected because the branch and bound thresholds are insensitive to the data density.

In the diagrams of Figure 15 each function examines a different search space around the query to retrieve the same number of users k. In the following experiment, we control the function parameters so that they explore the same search space, and consequently consider the same number of users. LC and HGS are based on range queries, and can therefore be explicitly set to cover a specified search space. For LC we use: $w = 0.5$, $C = 1$ to $5km$, and for HGS: $w = \{0.25, 0.5, 0.75, 1, 1.25\}$ (the search space is $4 \cdot w$). Since for GST and RC the search space cannot be defined explicitly, we adjust the value of w so that the query terminates with the user closest to the boundaries of the search space.



(a) Sparse (b) Dense

Fig. 16 Execution Time vs. Search Space (Real Data)

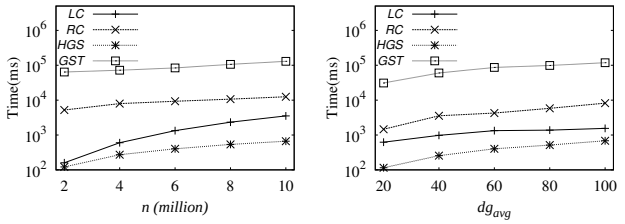


(a) Sparse (b) Dense

Fig. 17 Execution Time vs. Preference Setting (Real Data)

Figure 16 plots the running time in sparse and dense areas as a function of the range of the search space. HGS is the fastest algorithm for both sparse and dense areas, because it examines the minimum number of users, i.e., users within distance w from q and their friends whose distance to q does not exceed $4 \cdot w$. LC is the second fastest approach in sparse areas, but it is outperformed by RC in dense areas because the set intersection operations performed by LC become increasingly expensive as the number of users grows. Finally, GST is the most expensive algorithm since it takes into account the social connectivity of the result sets.

All functions involve a parameter w that can be used to adjust the trade-off between the social and the spatial aspects. However, the meaning and value range of w is different in each function. In order to study the effect of w on performance, we use the values of w shown in Tables 2, 3, 4 and 5, for LC, RC, HGS and GST, respectively. Figures 17(a) and 17(b) illustrate the running time in sparse and dense areas, for $k = 32$. The x-axis corresponds to the value of w at a particular row in the tables, e.g., the first value is 0.1, 0.1, 0.5 and 0.5 for LC, RC, HGS and GST. Since a large weight emphasizes social connectivity over spatial proximity, the search space, and the running time increase with w. LC exhibits the most significant impact because the search space is directly determined by w.



(a) Time vs. $n(dg_{avg} = 60)$ (b) Time vs. $dg_{avg}(n = 6M)$

Fig. 18 Execution Time vs. n and dg_{avg} (Synthetic Data, $k = 32$)

9.2 Synthetic Data

To evaluate the scalability of the algorithms, we used the method of [3] to construct synthetic GeoSNs of different user cardinality (n) and average degree (dg_{avg}). In particular, [3] generates a social graph using the Barabasi-Albert model [7]. Then, starting from a random user at a random location, it assigns locations to the users based on their distances to their friends, which are randomly derived from a power-law distribution[7]. The users are spread in an overall area of $13,500\ km^2$. The value of w is set to 0.5 in all ranking functions except GST[8], where $w = 0.01$.

Figure 18(a) plots the running time versus n, for $dg_{avg} = 60$ and $k = 32$. The relative performance of the algorithms is consistent with Figure 15, where HGS is the fastest method. The performance of LC deteriorates with the cardinality due to the higher number of users in the relevant range. On the other hand, as discussed in the previous subsection, RC and GST are not seriously affected by the data density.

Figure 18(b) measures the running time versus dg_{avg} ($n = 6M$ and $k = 32$). The impact of dg_{avg} on LC is minimal because the number of users within the relevant range is independent of the social connectivity. For other methods, the cost increases with dg_{avg} due to different reasons. In HGS, the number of candidate users who have friends in the initial range grows. For RC and GST, the maximum social degree (dg_{max}) and the number of triangles (tr_{max}) yield looser bounds. For instance, if $dg_{avg} = 20$, we have $dg_{max} = 2212$ and $tr_{max} = 2650$, while if $dg_{avg} = 80$, dg_{max} and tr_{max} are 3834 and 27475, respectively.

Summarizing, the most efficient top-k methods in our settings are HGS and LC, which often outperform the rest by orders of magnitude. RC has relatively low

[7] Analysis on real GeoSN datasets have shown that the distances between pairs of friends follow a power law [9].

[8] The Barabasi-Albert generator produces graphs with small expected number of triangles. Therefore, we assign high importance to the spatial proximity in order to avoid examining a large fraction of the dataset.

cost for high user density and small values of k. GST is consistently the most expensive algorithm, but the only one that explicitly considers connectivity.

10 Conclusion

With the proliferation of GPS-enabled devices and sophisticated location based services, ranking the users with respect to their position and friends near a query point is becoming increasingly important. In this paper, we present four ranking functions that cover a wide range of application requirements. For each function, we develop a specialized algorithm to retrieve the top-k users. We extensively evaluate each function on both the quality/characteristics of the results, and the efficiency of the associated algorithm.

In the future, we plan to focus on improved query processing algorithms, especially for functions, such as GST, which are expensive. Moreover, we intend to explore novel functions that may exhibit different characteristics, as well as their combination for meta-ranking; e.g., we could integrate HGS with another method in order to favor users whose friends are concentrated near the query. Finally, we plan to investigate the adaptation of the proposed methods to related application domains, such as spatial-keyword search in geo-social networks.

References

1. Foursquare statistics. `https://foursquare.com/about/`.
2. GroupOn Now! deals available on Foursquare. `https://blog.groupon.com/cities/groupon-now-deals-available-in-foursquare/`.
3. N. Armenatzoglou, S. Papadopoulos, and D. Papadias. A general framework for geo-social query processing. In *VLDB*, 2013.
4. F. Aurenhammer and H. Edelsbrunner. An optimal algorithm for constructing the weighted voronoi diagram in the plane. *Pattern Recognition*, 17(2):251–257, 1984.
5. M. Babenko, A. Goldberg, A. Gupta, and V. Nagarajan. Algorithms for hub label optimization. In *Automata, Languages, and Programming*, volume 7965 of *LNCS*, pages 69–80. 2013.
6. P. Babington. *The title of the work*, volume 4 of *10*. The name of the publisher, The address, 3 edition, 7 1993. An optional note.
7. S. Bornholdt, H. G. Schuster, and J. Wiley. *Handbook of graphs and networks*, volume 2. Wiley Online Library, 2003.
8. C. Brown, V. Nicosia, S. Scellato, A. Noulas, and C. Mascolo. Where online friends meet: Social communities in location-based networks. In *ICWSM*, 2012.
9. E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: user movement in location-based social networks. In *SIGKDD*, 2011.
10. G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. In *PVLDB*, 2009.
11. J. E. Hirsch. An index to quantify an individual's scientific research output. *PNAS*, 102(46):16569–16572, 2005.
12. X. Hu, Y. Tao, and C.-W. Chung. Massive graph triangulation. In *SIGMOD*, 2013.
13. M. G. Kendall. A new measure of rank correlation. *Biometrika*, pages 81–93, 1938.
14. W. Liu, W. Sun, C. Chen, Y. Huang, Y. Jing, and K. Chen. Circle of friend query in geo-social networks. In *DASFAA*, 2012.
15. K. Mouratidis, J. Li, Y. Tang, and N. Mamoulis. Joint search by social and spatial proximity. *Knowledge and Data Engineering, IEEE Transactions on*, 16(10):1169–1184, 2014.
16. S. Scellato, C. Mascolo, M. Musolesi, and V. Latora. Distance matters: Geo-social metrics for online social networks. In *WOSN*, 2010.
17. S. Scellato, A. Noulas, R. Lambiotte, and C. Mascolo. Socio-spatial properties of online location-based social networks. *ICWSM*, 11:329–336, 2011.
18. S. Scellato, A. Noulas, and C. Mascolo. Exploiting place features in link prediction on location-based social networks. In *SIGKDD*, 2011.
19. Y. Tao, J. Zhang, D. Papadias, and N. Mamoulis. An efficient cost model for optimization of nearest neighbor search in low and medium dimensional spaces. *Knowledge and Data Engineering, IEEE Transactions on*, 16(10):1169–1184, 2004.
20. Y. van Gennip, B. Hunter, R. Ahn, P. Elliott, K. Luh, M. Halvorson, S. Reid, M. Valasik, J. Wo, G. E. Tita, et al. Community detection using spectral clustering on sparse geosocial data. *SIAM Journal on Applied Mathematics*, 73(1):67–83, 2013.
21. H. Wang, M. Terrovitis, and N. Mamoulis. Location recommendation in location-based social networks using user check-in data. In *SIGSPATIAL*, pages 364–373, 2013.
22. D. Wu, M. Yiu, C. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, 2011.
23. D.-N. Yang, C.-Y. Shen, W.-C. Lee, and M.-S. Chen. On socio-spatial group query for location-based social networks. In *SIGKDD*, 2012.
24. M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *SIGIR*, 2011.
25. M. L. Yiu, L. H. U, S. Šaltenis, and K. Tzoumas. Efficient proximity detection among mobile users via self-tuning policies. In *PVLDB*, 2010.
26. C. Zhang, L. Shou, K. Chen, G. Chen, and Y. Bei. Evaluating geo-social influence in location-based social networks. In *CIKM*, 2012.
27. Y. Zheng, L. Zhang, Z. Ma, X. Xie, and W. Ma. Recommending friends and locations based on individual location history. *ACM Transactions on the Web (TWEB)*, 5(1):5, 2011.
28. Y. Zhou, X. Xie, C. Wang, Y. Gong, and W.-Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, 2005.